เรียนรู้
คอมพิวเตอร์กราฟิกส์ 2 มิติ
ด้วยภาษา C

พร้อมคำอธิบายและตัวอย่าง
โปรแกรมภาษาซี

- หลักการสร้างภาพกราฟิกส์
- หลักการออกแบบระบบกราฟิกส์
- การแปลงภาพกราฟิกส์
- การสร้างภาพเคลื่อนไหว



ตะวันเรืองศรี

ซีเอ็ด

เรียนรู้ คอมพิวเตอร์กราฟิกส์ 2 มิติ ด้วยภาษา C

สมพัฒน์ รุ่งตะวันเรืองศรี

ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์
มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่

บริษัท ซีเอ็ดยูเคชั่น จำกัด
SCIENCE, ENGINEERING & EDUCATION CO.,LTD.

ราคา 220 บาท

ISBN 974-509-551-6

1 3 2 – 0 0 0 – 2 2 8
0 5 3 3 3 2 1 0 9 8 7 6 5

จัดพิมพ์และจัดจำหน่ายโดย

บริษัท ซีเอ็ดยูเคชั่น จำกัด
SCIENCE, ENGINEERING & EDUCATION CO., LTD.

800/43-45 ซอยตระกูลสุข ถนนอโศกดินแดง เขตห้วยขวาง
กรุงเทพฯ 10400 โทร. 2486280-9 โทรสาร 2465999

พิมพ์ที่ หจก. เอช-เอน การพิมพ์ โทร. 2112622
402/16 ซอยดอนกุศล เขตยานนาวา กทม. 10120
นายประพันธ์ จิระมงคล ผู้พิมพ์ผู้โฆษณา พ.ศ. 2535

คำนำ

คอมพิวเตอร์กราฟิกส์ เป็นสาขาวิชาที่มีการศึกษาค้นคว้ากันอย่างกว้างขวางในต่างประเทศ ผลงานต่าง ๆ ทางด้านคอมพิวเตอร์กราฟิกส์ก็ปรากฏให้เห็นอยู่ทั่วไปในทุกวงการ ที่เห็นได้ชัดก็ได้แก่ วงการโฆษณาและวงการภาพยนตร์ ภาพแปลก ๆ มากมายที่ปรากฏสร้างความรู้สึกประหลาดใจให้กับผู้พบเห็น นอกจากนี้ การใช้งานคอมพิวเตอร์กราฟิกส์ยังแทรกซึมไปเกือบจะทุกวงการแล้ว ไม่ว่าจะเป็นด้านการศึกษา การแพทย์ การเสริมสวย และอื่น ๆ อีกมากมาย ดังนั้นเราจึงควรเรียนรู้ทำความเข้าใจในเรื่องคอมพิวเตอร์กราฟิกส์ไว้บ้าง

คอมพิวเตอร์กราฟิกส์เป็นสาขาวิชาซึ่งดูเหมือนค่อนข้างใหม่ในบ้านเรา โดยสังเกตได้จากตลาดหนังสือวิชาการภาษาไทย ซึ่งแทบจะไม่มีหนังสือที่เกี่ยวกับคอมพิวเตอร์กราฟิกส์โดยตรงเลย ทำให้ผู้เขียนเกิดความคิดที่จะเขียนหนังสือประเภทนี้ขึ้นมา

เนื้อหาทั้งหมดในหนังสือเล่มนี้เรียบเรียะขึ้นจากการศึกษาของผู้เขียนเอง โดยได้ศึกษาจากหนังสือเกี่ยวกับคอมพิวเตอร์กราฟิกส์ภาษาต่างประเทศหลายเล่ม ซึ่งได้อ้างอิงไว้ท้ายเล่มแล้วเนื้อหาเป็นความรู้เบื้องต้นที่เราควรรู้เกี่ยวกับคอมพิวเตอร์กราฟิกส์ 2 มิติ พร้อมทั้งมีโปรแกรมตัวอย่างโดยใช้ภาษาซี คอมไพเลอร์ภาษาซีที่ใช้คือ Turbo C++ เวอร์ชั่น 1.00 ภายในตัวโปรแกรมหลัก จะเป็นคำสั่งซึ่งมีกำหนดไว้ใน ANSI C. ทั้งหมด โดยหลีกเลี่ยงการใช้คำสั่งซึ่งมีเฉพาะในเทอร์โบซี เพื่อทำให้โปรแกรมไม่ผูกติดกับเทอร์โบซี การหลีกเลี่ยงใช้คำสั่งที่มีเฉพาะในเทอร์โบซีทำได้โดยกำหนดคำสั่งนั้นให้เป็นชื่อฟังก์ชันใหม่ซึ่งมีคำสั่งนี้อยู่ภายในฟังก์ชัน การทำเช่นนี้จะช่วยให้ผู้ที่ใช้คอมไพเลอร์ภาษาซีตัวอื่น ๆ สามารถใช้โปรแกรมตัวอย่างเหล่านี้ได้ด้วย

โดยทำการแก้ไขโปรแกรมเพียงเปลี่ยนคำสั่งภายในฟังก์ชันให้เป็นคำสั่งที่สอดคล้องกับคำสั่งที่มีใช้ในคอมไพเลอร์ภาษาซีซึ่งใช้อยู่เท่านั้น

สุดท้ายนี้ ผู้เขียนหวังว่าหนังสือเล่มนี้จะเป็นประโยชน์ต่อผู้อ่านพอสมควร

สมพัฒน์ รุ่งตะวันเรืองศรี

ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์
มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่

สารบัญ

1 คอมพิวเตอร์กราฟิกส์

1.1 บทนำ

คอมพิวเตอร์กราฟิกส์เป็นคำที่หมายถึง การสร้างและการจัดการกับภาพกราฟิกส์โดยใช้คอมพิวเตอร์ คำจำกัดความสั้น ๆ นี้ไม่ได้อธิบายถึงเหตุผลของการพัฒนาคอมพิวเตอร์กราฟิกส์ ซึ่งเริ่มต้นมาจากการเป็นเทคนิคอย่างหนึ่งในการแสดงข้อมูลตัวเลขจำนวนมาก ๆ ให้อยู่ในรูปที่ชัดเจนกว่าเดิมและทำความเข้าใจได้ง่ายกว่าเดิม เช่น ข้อมูลอาจแสดงได้ในรูปของ เส้นกราฟ แผนภาพ แผนภูมิ แทนที่จะเป็นตารางของตัวเลข เป็นต้น จากนั้นมาการใช้ภาพกราฟิกส์แสดงผลแทนข้อมูลหรือข่าวสารที่ยุ่งยากก็มีการพัฒนามากขึ้นเรื่อย ๆ จนในปัจจุบัน มีการใช้ภาพกราฟิกส์ในงานทุก ๆ ด้าน ไม่ว่าจะเป็นทางด้านธุรกิจ โรงงานอุตสาหกรรม งานศิลปะ งานด้านการบันเทิง งานโฆษณา งานด้านการศึกษา การวิจัย การฝึกอบรม และงานทางการแพทย์ จะเห็นได้ชัดเจนว่าสาขาวิชาทางคอมพิวเตอร์กราฟิกส์นั้นเริ่มมีความสำคัญมากขึ้นทุกที เนื่องจากเป็นเครื่องมือที่สามารถช่วยงานได้เป็นอย่างมาก

1.2 ประวัติโดยย่อเกี่ยวกับคอมพิวเตอร์กราฟิกส์

ในปี ค.ศ. 1940 คอมพิวเตอร์จะแสดงภาพกราฟิกส์โดยใช้เครื่องพิมพ์ โดยรูปภาพที่ได้จะเป็นภาพที่เกิดจากการใช้ตัวอักษรมาประกอบกัน ดังตัวอย่างในรูปที่ 1.1

ในปี ค.ศ. 1950 สถาบันเทคโนโลยีแห่งแมสซาชูเซสต์ (MIT) ได้พัฒนาคอมพิวเตอร์ซึ่งมีหลอดภาพ CRT (cathode ray tube) เป็นส่วนแสดงผลแทนเครื่องพิมพ์ เนื่องจากมีความต้องการที่จะให้การติดต่อระหว่างผู้ใช้กับเครื่องคอมพิวเตอร์มีความรวดเร็วยิ่งขึ้น

```
                  1111111         1111111
                1111111111       111111111111
               1111111111111   111111111111111
              111111111111111 111111111111111111
==            111111111111111111111111111111111
 ===========================111111111111111111=======>
==              1111111111111111111111111111
                  11111111111111111111111
                     11111111111111111
                        11111111111
                           111111
                             111
```

รูปที่ 1.1 การแสดงภาพโดยใช้ตัวอักษรมาประกอบกัน

ในปี ค.ศ. 1950 ระบบ SAGE ของกองทัพอากาศสหรัฐอเมริกาสามารถแปลงสัญญาณจากเรดาร์ให้เป็นภาพบนจอคอมพิวเตอร์ได้ ระบบนี้เป็นระบบกราฟิกส์เครื่องแรกที่ใช้ปากกาแสง (light pen : อุปกรณ์สำหรับรับข้อมูลชนิดหนึ่ง) สำหรับการเลือกสัญลักษณ์บนจอภาพได้

ในช่วงปี ค.ศ. 1950-1960 มีการทำวิจัยเรื่องเกี่ยวกับระบบคอมพิวเตอร์เป็นจำนวนมาก ซึ่งต่อมาได้กลายเป็นต้นแบบของระบบคอมพิวเตอร์กราฟิกส์สมัยใหม่ ตัวอย่างเช่น ในปี ค.ศ. 1963 วิทยานิพนธ์ปริญญาเอกของ อีวาน ซูเธอร์แลนด์ (Ivan Sutherland) เป็นการพัฒนาระบบการวาดเส้น ซึ่งผู้ใช้สามารถกำหนดจุดบนจอภาพได้โดยตรงโดยการใช้ปากกาแสง จากนั้นระบบกราฟิกส์จะสามารถลากเส้นเชื่อมจุดต่าง ๆ เหล่านี้เข้าด้วยกัน กลายเป็นภาพโครงสร้างรูปหลายเหลี่ยม ระบบนี้ได้กลายเป็นหลักการพื้นฐานของโปรแกรมช่วยในการออกแบบระบบงานต่าง ๆ เช่น การออกแบบระบบไฟฟ้า และการออกแบบเครื่องจักร เป็นต้น

ในระบบหลอดภาพ CRT สมัยแรกนั้น เราสามารถวาดเส้นตรงระหว่างจุดสองจุดบนจอภาพได้ แต่ภาพเส้นที่วาดจะจางหายไปจากจอภาพอย่างรวดเร็ว จึงต้องมีการวาดซ้ำลงที่เดิมหลาย ๆ ครั้งในหนึ่งวินาที เพื่อให้เราสามารถมองเห็นว่าเส้นไม่จางหายไป ซึ่งระบบแบบนี้มีราคาแพงมากในช่วงต้นปี ค.ศ. 1960 แต่ต่อมาในปี ค.ศ. 1965 จึงมีราคาถูกลงเนื่องจากบริษัทไอบีเอ็ม (IBM)

ได้ผลิตออกมาขายเป็นจำนวนมากในราคาเครื่องละ 100,000 ดอลลาร์ จากการที่ราคาของจอภาพถูกลงมากนี่เอง ทำให้สาขาคอมพิวเตอร์กราฟิกส์เริ่มเป็นที่สนใจของคนทั่วไป

ในปี ค.ศ. 1968 บริษัทเทคโทรนิกส์ (Tektronix) ได้ผลิตจอภาพแบบเก็บภาพไว้ได้จนกว่าต้องการจะลบ (storage – tube CRT) ซึ่งระบบนี้ไม่ต้องการหน่วยความจำและระบบการวาดซ้ำ จึงทำให้มีราคาถูกลงมาก บริษัทตั้งราคาขายไว้เพียง 15,000 ดอลล่าร์เท่านั้น จอภาพแบบนี้จึงเป็นที่นิยมกันมากในช่วงเวลา 5 ปีต่อมา

กลางปี ค.ศ. 1970 เป็นช่วงเวลาที่อุปกรณ์ทางคอมพิวเตอร์เริ่มมีราคาลดลงมาก ทำให้ฮาร์ดแวร์ของระบบคอมพิวเตอร์กราฟิกส์มีราคาถูกลงตามไปด้วย ผู้ใช้ทั่วไปจึงสามารถนำมาใช้ในงานของตนได้ ทำให้การใช้คอมพิวเตอร์กราฟิกส์เริ่มแพร่หลายไปในงานด้านต่าง ๆ มากขึ้น

สำหรับซอฟต์แวร์ทางด้านกราฟิกส์ก็มีการพัฒนาควบคู่มากับฮาร์ดแวร์เช่นกัน ซึ่งมีการเริ่มต้นจาก อีวาน ซูเธอร์แลนด์ ผู้ซึ่งได้ออกแบบวิธีการหลัก ๆ รวมทั้งโครงสร้างข้อมูลของระบบคอมพิวเตอร์กราฟิกส์ ต่อมาก็มี สตีเฟน คูน (Steven Coons, 1966) และ ปิแอร์ เบเซอร์ (Pierre Bazier, 1972) ซึ่งศึกษาเกี่ยวกับการสร้างเส้นโค้งและภาพพื้นผิว ทำให้ปัจจุบันเราสามารถสร้างภาพ 3 มิติได้สมจริงสมจังมากขึ้น ในช่วง 10 ปีต่อมาได้มีการพัฒนาวิธีการสร้างภาพมากมายสำหรับใช้ในระบบคอมพิวเตอร์กราฟิกส์ และปัจจุบันเราก็ได้เห็นผลงานที่สวยงามและแปลกตาซึ่งเป็นผลจากการศึกษาวิจัยต่าง ๆ ในอดีตนั่นเอง

1.3 คอมพิวเตอร์กราฟิกส์กับการประยุกต์ใช้ในงานด้านต่าง ๆ

1.3.1 คอมพิวเตอร์กราฟิกส์กับการออกแบบ

คอมพิวเตอร์กราฟิกส์ได้ถูกนำมาใช้ในงานออกแบบมาเป็นเวลานาน เราคงจะเคยได้ยินคำว่า CAD (computer-aided design) ซึ่งเป็นโปรแกรมสำหรับช่วยในการออกแบบทางวิศวกรรม โปรแกรมเหล่านี้จะช่วยให้ผู้ออกแบบหรือวิศวกรออกแบบงานต่าง ๆ ได้สะดวกขึ้น กล่าวคือ ผู้ออกแบบสามารถเขียนเป็นแบบลายเส้นแล้วลงสี แสงเงา เพื่อให้ดูคล้ายกับของจริงได้ นอกจากนี้แล้ว เมื่อผู้ออกแบบกำหนดขนาดของวัตถุลงในระบบ CAD แล้ว ผู้ออกแบบยังสามารถย่อหรือขยายภาพนั้น หรือต้องการหมุนภาพไปในมุมต่าง ๆ ได้ด้วย การแก้ไขแบบก็ทำได้ง่ายและสะดวกกว่าการออกแบบบนกระดาษ

ทางด้านวิศวกรรมไฟฟ้าและอิเล็กทรอนิกส์ คอมพิวเตอร์กราฟิกส์ถูกนำมาใช้ในการออกแบบวงจรต่าง ๆ ผู้ออกแบบสามารถวาดวงจรบนจอภาพโดยใช้สัญลักษณ์ต่าง ๆ ที่ระบบจัดเตรียมไว้ให้แล้วมาประกอบกันเป็นวงจรที่ต้องการ ผู้ออกแบบสามารถแก้ไข ตัดต่อ เพิ่มเติมวงจรได้โดย

สะดวก นอกจากนี้ยังมีโปรแกรมสำหรับออกแบบ PCB (printed circuit board) ซึ่งมีความสามารถจัดการให้แผ่นปรินต์มีขนาดที่จะวางอุปกรณ์ทางอิเล็กทรอนิกส์ได้เหมาะสมที่สุด

การออกแบบพาหนะต่าง ๆ เช่น รถยนต์ เครื่องบิน หรือเครื่องจักรต่าง ๆ ในปัจจุบันก็ใช้ระบบ CAD นักออกแบบสามารถจะออกแบบส่วนย่อย ๆ แต่ละส่วนก่อน แล้วนำมาประกอบกันเป็นส่วนใหญ่ขึ้นจนเป็นเครื่องจักรเครื่องยนต์ที่ต้องการได้ นอกจากนี้ในบางระบบยังสามารถที่จะทดสอบแบบจำลองที่ออกแบบไว้ได้ด้วย เช่น อาจจะออกแบบรถยนต์แล้วนำโครงสร้างของรถที่ออกแบบนั้นมาจำลองการวิ่ง โดยให้วิ่งที่ความเร็วต่าง ๆ กันแล้วตรวจดูผลที่ได้ ซึ่งการทดลองแบบนี้สามารถทำได้ในระบบคอมพิวเตอร์และจะประหยัดกว่าการสร้างรถจริง ๆ แล้วนำออกมาศึกษาทดสอบการวิ่ง

การออกแบบโครงสร้าง เช่น ตึก บ้าน สะพาน หรือโครงสร้างใด ๆ ทางวิศวกรรมโยธาและสถาปัตยกรรม ก็สามารถทำได้โดยใช้ CAD ช่วยในการออกแบบ หลังจากสถาปนิกออกแบบโครงสร้างในแบบ 2 มิติเสร็จแล้ว ระบบ CAD สามารถจัดการให้เป็นภาพ 3 มิติและยังสามารถแสดงภาพที่มุมมองต่าง ๆ กันได้ตามที่ผู้ออกแบบต้องการ นอกจากนี้ในบางระบบสามารถแสดงภาพให้ปรากฏต่อผู้ออกแบบราวกับว่าผู้ออกแบบสามารถเดินเข้าไปภายในอาคารที่ออกแบบได้ด้วย

1.3.2 กราฟและแผนภาพ

คอมพิวเตอร์กราฟิกส์ถูกนำมาใช้ในการแสดงภาพกราฟและแผนภาพของข้อมูลได้เป็นอย่างดี โปรแกรมทางกราฟิกส์ทั่วไปในท้องตลาดจะเป็นโปรแกรมที่ใช้ในการสร้างภาพกราฟและแผนภาพ โปรแกรมเหล่านี้สามารถสร้างกราฟได้หลายแบบ เช่น กราฟเส้น กราฟแท่ง และกราฟวงกลม นอกจากนี้ยังสามารถแสดงภาพกราฟได้ทั้งในรูปแบบ 2 มิติและ 3 มิติ ทำให้ภาพกราฟที่ได้ดูดีและน่าสนใจ กราฟและแผนภาพทางธุรกิจ เช่น กราฟหรือแผนภาพแสดงการเงิน สถิติและข้อมูลทางเศรษฐกิจ จะเป็นประโยชน์ต่อผู้บริหารหรือผู้จัดการกิจการมาก เนื่องจากสามารถทำความเข้าใจกับข้อมูลได้ง่ายและรวดเร็วกว่าเดิม ในงานวิจัยต่าง ๆ เช่น การศึกษาทางฟิสิกส์ กราฟและแผนภาพมีส่วนช่วยให้นักวิจัยทำความเข้าใจกับข้อมูลได้ง่ายขึ้นเมื่อข้อมูลที่ต้องวิเคราะห์มีจำนวนมาก

ระบบข้อมูลทางภูมิศาสตร์ หรือ GIS (geographical information system) ก็เป็นรูปแบบหนึ่งของการแสดงข้อมูลในทำนองเดียวกับกราฟและแผนภาพ ข้อมูลทางภูมิศาสตร์จะถูกเก็บลงในระบบคอมพิวเตอร์ แล้วให้ระบบคอมพิวเตอร์กราฟิกส์จัดการแสดงข้อมูลเหล่านั้นออกมาทางจอภาพในรูปของแผนที่ทางภูมิศาสตร์

1.3.3 ภาพศิลป์โดยคอมพิวเตอร์กราฟิกส์

การวาดภาพในปัจจุบันนี้ใคร ๆ ก็สามารถวาดได้แล้วโดยไม่ต้องใช้พู่กันกับจานสี แต่จะใช้คอมพิวเตอร์กราฟิกส์แทน ภาพที่วาดในระบบคอมพิวเตอร์กราฟิกส์นี้เราสามารถกำหนดสี แสง เงา รูปแบบลายเส้นที่ต้องการได้โดยง่าย ภาพโฆษณาทางโทรทัศน์หลายชิ้นก็เป็นงานจากการใช้คอมพิวเตอร์กราฟิกส์ ข้อดีของการใช้คอมพิวเตอร์วาดภาพก็คือ เราสามารถแก้ไขเพิ่มเติมส่วนที่ต้องการได้ง่าย นอกจากนี้เรายังสามารถนำภาพต่าง ๆ เก็บในระบบคอมพิวเตอร์ได้โดยใช้เครื่องสแกนเนอร์ (scanner) แล้วนำภาพเหล่านั้นมาแก้ไข

1.3.4 ภาพเคลื่อนไหวโดยใช้คอมพิวเตอร์

ภาพยนตร์การ์ตูนและภาพยนตร์ประเภทนิยายวิทยาศาสตร์หรือภาพยนตร์ที่ใช้เทคนิคพิเศษต่าง ๆ ในปัจจุบันมีการนำคอมพิวเตอร์กราฟิกส์เข้ามาช่วยในการออกแบบและสร้างภาพเคลื่อนไหว (computer animation) มากขึ้น เนื่องจากเป็นวิธีที่สะดวก รวดเร็ว และง่ายกว่าวิธีอื่น ๆ นอกจากนี้ภาพที่ได้ยังดูสมจริงมากขึ้น เช่น ภาพยานอวกาศที่ปรากฏในภาพยนตร์ประเภทนิยายวิทยาศาสตร์ เป็นต้น การใช้คอมพิวเตอร์กราฟิกส์ช่วยให้ภาพที่อยู่ในจินตนาการของมนุษย์สามารถนำออกมาทำให้ปรากฏเป็นจริงได้

ภาพเคลื่อนไหวมีประโยชน์มากทั้งในระบบการศึกษา การอบรม การวิจัย และการจำลองการทำงาน เช่น จำลองการขับรถ การขับเครื่องบิน เป็นต้น เกมคอมพิวเตอร์หรือวิดีโอเกมก็ใช้หลักการทำภาพเคลื่อนไหวในคอมพิวเตอร์กราฟิกส์เช่นกัน

1.3.5 อิเมจโปรเซสซิงก์

คำว่าอิเมจโปรเซสซิงก์ (image processing) หมายถึงการแสดงภาพที่เกิดจากการถ่ายรูปหรือจากการสแกนภาพให้ปรากฏบนจอภาพคอมพิวเตอร์ วิธีการทางอิเมจโปรเซสซิงก์จะต่างกับวิธีการของคอมพิวเตอร์กราฟิกส์ กล่าวคือ ในระบบคอมพิวเตอร์กราฟิกส์ ตัวคอมพิวเตอร์เองจะเป็นตัวที่สร้างภาพ แต่เทคนิคทางอิเมจโปรเซสซิงก์นั้นใช้คอมพิวเตอร์สำหรับจัดการรูปแบบของสีและแสงเงาที่มีอยู่แล้วในภาพให้เป็นข้อมูลทางดิจิตอล แล้วอาจจะมีวิธีการทำให้ภาพที่รับเข้ามานั้นมีความชัดเจนมากขึ้นก่อน จากนั้นก็จัดการกับข้อมูลดิจิตอลนี้ให้เป็นภาพส่งออกไปที่จอภาพของคอมพิวเตอร์อีกที วิธีการนี้มีประโยชน์ในการแสดงภาพของวัตถุที่เราไม่สามารถจะเห็นได้โดยตรง เช่น ภาพถ่ายดาวเทียม ภาพจากทีวีสแกนของหุ่นยนต์อุตสาหกรรม เป็นต้น

เมื่อภาพถ่ายถูกทำให้เป็นข้อมูลดิจิตอลแล้ว เราก็สามารถจะจัดการแก้ไขเปลี่ยนแปลงภาพนั้นได้โดยจัดการกับข้อมูลดิจิตอลของภาพนั่นเอง ซึ่งเราก็จะใช้หลักการของคอมพิวเตอร์

กราฟิกส์มาใช้กับข้อมูลเหล่านี้ได้ เช่น ในภาพสำหรับการโฆษณา เราสามารถทำให้ภาพที่เห็นเหมือนภาพถ่ายนั้นแปลกออกไปจากเดิมได้โดยการมีภาพบางอย่างเพิ่มเข้าไปหรือบางส่วนของภาพนั้นหายไป ทำให้เกิดเป็นภาพที่ไม่น่าจะเป็นจริงแต่ดูเหมือนกับเกิดขึ้นจริงได้ เป็นต้น

เทคนิคของอิเมจโปรเซสซิงก์สามารถประยุกต์ใช้กับการแพทย์ได้ เช่น เครื่องเอกซเรย์โทโมกราฟี (X-ray tomography) ซึ่งใช้สำหรับแสดงภาพตัดขวางของระบบร่างกายมนุษย์ เป็นต้น

จากที่กล่าวมาแล้ว เราจะเห็นได้ว่าคอมพิวเตอร์กราฟิกส์นั้นนับวันก็ยิ่งมีความสำคัญในสาขาวิชาต่าง ๆ มากขึ้น ดังนั้นจึงเป็นการดีที่เราควรจะมีความรู้ความเข้าใจในหลักการและเทคนิคเบื้องต้นต่าง ๆ ที่ใช้ในคอมพิวเตอร์กราฟิกส์

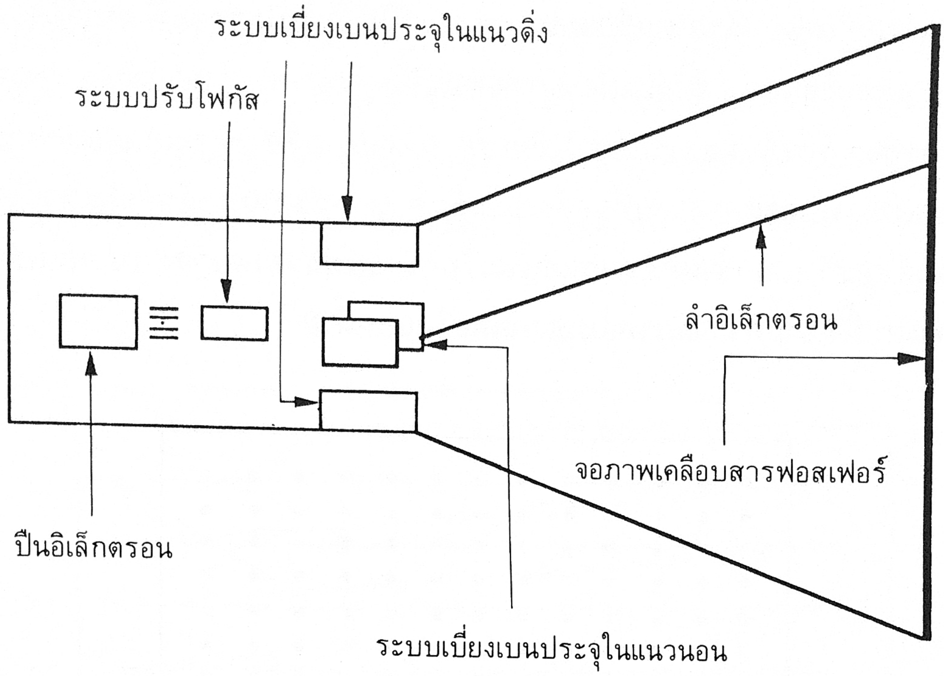
1.4 ระบบกราฟิกส์แบบอินเตอร์แอกตีฟ

เมื่อเราพิมพ์ภาพกราฟิกส์ที่สร้างขึ้น มันไม่จำเป็นนักที่จะต้องให้ภาพนั้นปรากฏทันทีบนกระดาษ กราฟิกส์แบบนี้เรียกกันว่า *กราฟิกส์แบบสถิต (static graphics)* แต่ถ้าเป็นการเล่นวิดีโอเกม ถ้าเวลาในการตอบสนองมากกว่า 1/10 วินาทีหลังจากการเคลื่อนที่ของจอยสติ๊กก็อาจจะไม่เป็นที่ยอมรับได้ เนื่องจากในการเล่นนี้เราต้องการการตอบสนองแบบทันทีทันใด ภาพกราฟิกส์ซึ่งมีการเปลี่ยนแปลงตลอดเวลาแบบนี้จะต้องเปลี่ยนแปลงได้เร็วพอที่ผู้ใช้จะสามารถควบคุมภาพได้ ระบบแบบนี้เรียกว่า *ระบบกราฟิกส์แบบอินเตอร์แอกตีฟ (interactive graphics system)* สำหรับระบบแบบนี้ต้องการฮาร์ดแวร์ที่พิเศษซึ่งจะจัดการเฉพาะในเรื่องที่เกี่ยวกับการแสดงภาพและการตอบสนองผู้ใช้ ยิ่งภาพที่มีความซับซ้อนเหมือนจริงมากขึ้นเท่าใด การที่จะทำให้ภาพมีการเปลี่ยนแปลงอย่างรวดเร็วก็ทำได้ยากยิ่งขึ้นเท่านั้น

ระบบกราฟิกส์แบบอินเตอร์แอกตีฟ โดยทั่วไปจะประกอบด้วยส่วนสำคัญ 4 ส่วน คือ คอมพิวเตอร์, จอภาพสำหรับการแสดงภาพ, อุปกรณ์รับคำสั่งและข้อมูลจากผู้ใช้และอุปกรณ์สำหรับการพิมพ์ภาพ ซึ่งต่อไปจะได้กล่าวถึงรายละเอียดของส่วนประกอบแต่ละส่วน

1.4.1 จอภาพสำหรับการแสดงภาพ

จอภาพส่วนมากที่ใช้กันจะเป็นจอภาพชนิดเดียวกันกับจอภาพของโทรทัศน์ซึ่งเรียกกันว่า CRT ในรูปที่ 1.2 แสดงส่วนประกอบสำคัญของ CRT ซึ่งได้แก่ ปืนอิเล็กตรอน (electron gun) ซึ่งเมื่อร้อนจะปล่อยประจุลบออกมา ประจุลบเหล่านี้จะวิ่งไปหาประจุบวกซึ่งอยู่ที่จอภาพที่ฉาบด้วยสารฟอสเฟอร์ ระหว่างที่ประจุลบวิ่งไปนั้นจะต้องผ่านระบบปรับโฟกัสและระบบเบี่ยงเบนประจุซึ่งเป็นตัวบังคับให้ประจุลบวิ่งไปกระทบจอในตำแหน่งที่ต้องการได้



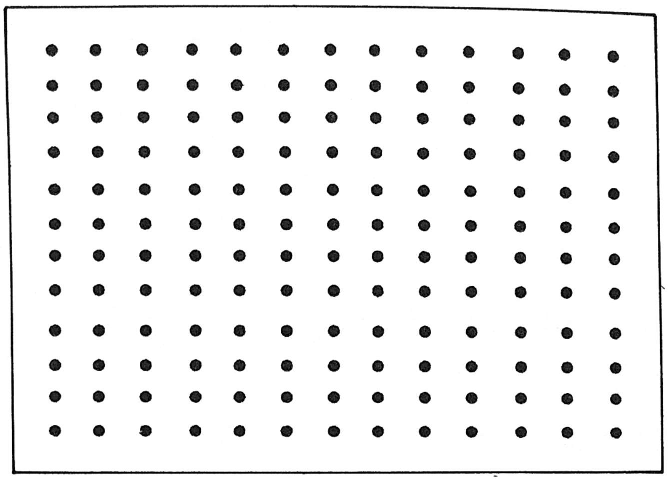
รูปที่ 1.2 ส่วนประกอบสำคัญของ CRT

ระบบปรับโฟกัสจะใช้สำหรับปรับลำประจุลบเพื่อเวลาที่ประจุลบกระทบกับจอภาพแล้วจะทำให้เกิดจุดสว่างเล็ก ๆ บนจอภาพ ส่วนระบบเบี่ยงเบนประจุจะประกอบด้วยแผ่นโลหะ 2 ชุด (สำหรับการเบี่ยงเบนในแนวนอนและการเบี่ยงเบนในแนวดิ่ง) ใช้สำหรับปรับทิศทางการวิ่งของประจุลบเพื่อให้กระทบที่ตำแหน่งต่าง ๆ ทุกตำแหน่งของจอภาพได้

เมื่อประจุลบนี้วิ่งกระทบจอภาพ สารฟอสเฟอร์ที่ฉาบอยู่บนจอภาพก็จะเปล่งแสงที่ตาคนมองเห็นได้ออกมา ความเข้มของแสงจะขึ้นอยู่กับจำนวนประจุลบที่วิ่งมาชน ส่วนที่เป็นสีดำบนจอภาพนั้นก็คือส่วนที่ไม่มีประจุลบหรือมีน้อยมากวิ่งไปชน แสงที่เกิดขึ้นบนจอภาพจะคงอยู่ได้ชั่วระยะเวลาเพียงเศษเสี้ยวของวินาทีเท่านั้น ดังนั้นเพื่อให้เราสามารถเห็นภาพได้ต่อเนื่องโดยไม่เกิดการกะพริบจะต้องมีการยิงประจุลบซ้ำที่เดิมนี้หลายครั้งใน 1 วินาที เราจะเรียกจอภาพประเภทนี้ว่า *รีเฟรชซีอาร์ที (refresh CRT)* จอภาพประเภทนี้ยังแบ่งออกเป็น 2 แบบ คือ แบบ *แรสเตอร์สแกน (raster scan)* และแบบ *แรนดอมเวกเตอร์ (random vector)* ถึงแม้ว่าจะมีการใช้งานจอภาพทั้งสองแบบแต่โดยทั่วไปแล้วมักจะใช้จอภาพแบบแรสเตอร์สแกนมากกว่า เนื่องจากเป็นจอภาพที่เราสามารถจัดการเกี่ยวกับการให้สีและแสงเงาได้ง่ายกว่าจอภาพแบบแรนดอมเวกเตอร์

สำหรับจอภาพสีก็ใช้หลอดภาพ CRT เช่นกัน แต่มีส่วนประกอบเพิ่มเติม กล่าวคือ จะมีปืนอิเล็กตรอน 3 ชุดสำหรับแม่สีแสงคือ สีแดง สีเขียว และสีน้ำเงิน บนจอภาพจะฉาบด้วยสารฟอสเฟอร์สามสีต่อหนึ่งจุด ซึ่งจัดวางเป็นรูปสามเหลี่ยม และจะมีระบบควบคุมอื่น ๆ เพื่อจัดการให้จอภาพสามารถแสดงสีต่าง ๆ ได้ตามต้องการ

(ก) จอภาพแบบแรสเตอร์สแกน จอภาพที่ใช้กับไมโครคอมพิวเตอร์โดยทั่วไปจะเป็นจอภาพแบบแรสเตอร์สแกน ซึ่งจะแบ่งจอภาพออกเป็นจุดเล็ก ๆ จำนวนมาก จุดเหล่านี้จะเป็นส่วนประกอบของภาพที่เล็กที่สุดเรียกว่า *พิกเซล (pixels หรือ picture elements)* จุดเหล่านี้จะจัดเรียงกันเป็นแบบตะแกรงโดยที่จุดตัดของเส้นตามแนวนอนกับเส้นตามแนวดิ่งก็คือหนึ่งจุดนั่นเอง (แสดงดังรูปที่ 1.3) เส้นตามแนวนอนจะเรียกว่า *เส้นแรสเตอร์สแกน (raster-scan lines)* ดังนั้นจอภาพแบบนี้จึงเรียกว่าจอภาพแบบแรสเตอร์สแกนด้วย

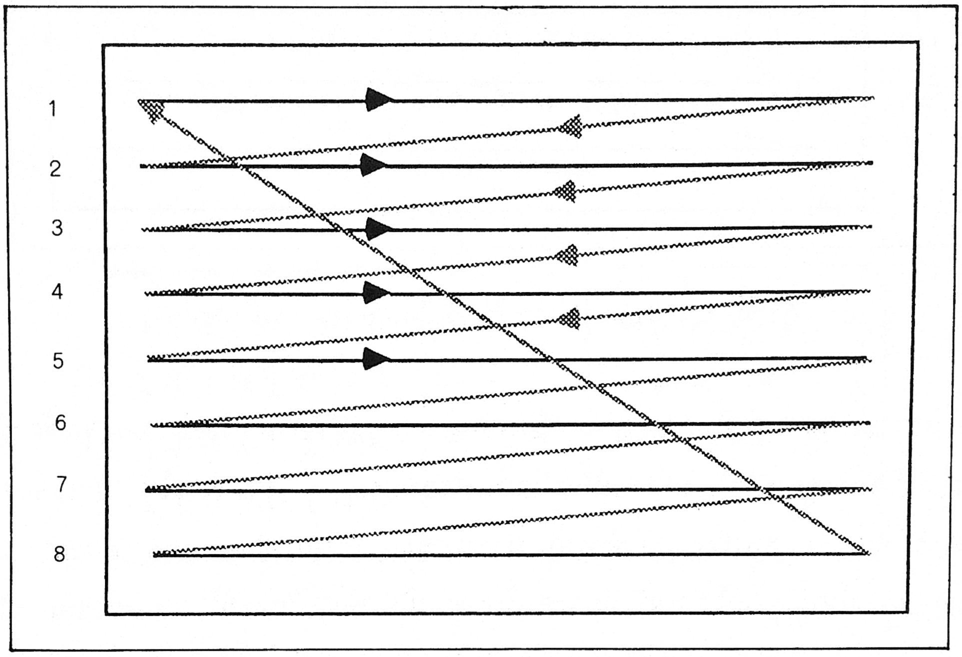


รูปที่ 1.3 ลักษณะจอภาพของจอภาพแบบแรสเตอร์สแกน จุดแต่ละจุดก็คือหนึ่งพิกเซล (ในความเป็นจริงจุดเหล่านี้จะอยู่ชิดกันมาก)

คุณภาพของจอภาพแบบแรสเตอร์อธิบายได้โดยความละเอียดของจอภาพ (resolution) ซึ่งก็คือจำนวนพิกเซลในหนึ่งเส้นสแกน (scan line) กับจำนวนเส้นสแกนที่มีบนจอภาพทั้งหมด ยิ่งความละเอียดของจอภาพมีมากเท่าใดก็ยิ่งแสดงภาพได้ละเอียดมากขึ้นเท่านั้น สำหรับจอภาพที่มีความละเอียดต่ำ (low-resolution) จะมีเส้นสแกนประมาณ 300 เส้น แต่ละเส้นสแกนจะมีพิกเซลประมาณ 400 พิกเซล ส่วนจอภาพที่มีความละเอียดสูง (high-resolution) จะมีเส้นสแกนอย่างต่ำ 1,000 เส้น และแต่ละเส้นจะมีพิกเซลมากกว่า 1,000 พิกเซล

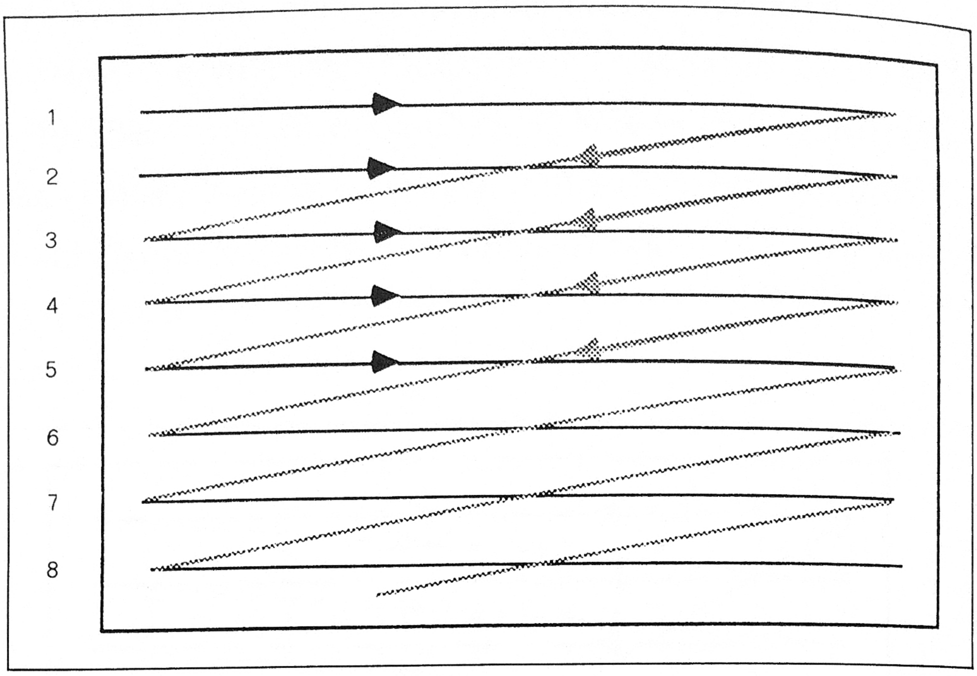
การยิงประจุลบไปยังจอภาพเพื่อให้เกิดเป็นจุดสว่างนั้นจะมีรูปแบบการทำงานที่แน่นอน (แสดงดังรูปที่ 1.4) กล่าวคือ จุดสว่างจะเริ่มเกิดที่มุมบนซ้ายของจอภาพก่อนเพราะจะมีการยิงประจุลบที่ตำแหน่งนี้เป็นตำแหน่งเริ่มต้นเสมอ จากนั้นก็จะยิงประจุลบไปตามแนวของเส้นสแกนจากซ้ายไปขวาจนกระทั่งถึงจุดทางขวาสุด ขั้นตอนนี้ก็จะเกิดแสงเปล่งออกมาเป็นเส้นตามแนวของเส้นสแกนหลังจากนั้นก็จะเริ่มยิงประจุลบไปยังจุดซ้ายสุดของจอภาพบนเส้นสแกนเส้นที่สองและยิงประจุลบไปตามแนวเส้นสแกนนี้จนถึงจุดขวาสุด และจะทำเช่นนี้ไปจนกว่าจะถึงจุดที่อยู่ที่ตำแหน่งมุมล่างขวาของจอภาพซึ่งอยู่บนเส้นสแกนเส้นสุดท้ายแล้วก็จะกลับไปเริ่มกระบวนการยิงประจุลบ

ใหม่ตามรูปแบบเดิม ซึ่งจะกระทำเช่นนี้หลายครั้งใน 1 วินาที จำนวนครั้งที่มีการยิงประจุลบได้ครบรอบดังกล่าวมาแล้วในช่วงเวลา 1 วินาทีจะเรียกว่า *อัตรารีเฟรช (refresh rate)* สำหรับส่วนกลับของอัตรารีเฟรช ซึ่งก็คือเวลาที่ใช้ในการแสดงภาพหนึ่งจอภาพจะเรียกว่า *เวลาเฟรม (frame time)* ถึงแม้ว่าพิกเซลที่เกิดขึ้นจะมีการจางหายไปตลอดเวลา แต่คนเราไม่สามารถมองเห็นการเปลี่ยนแปลงนี้ได้ เนื่องจากอัตรารีเฟรชมีค่ามากเพียงพอจึงทำให้คนเราเห็นภาพได้อย่างต่อเนื่อง



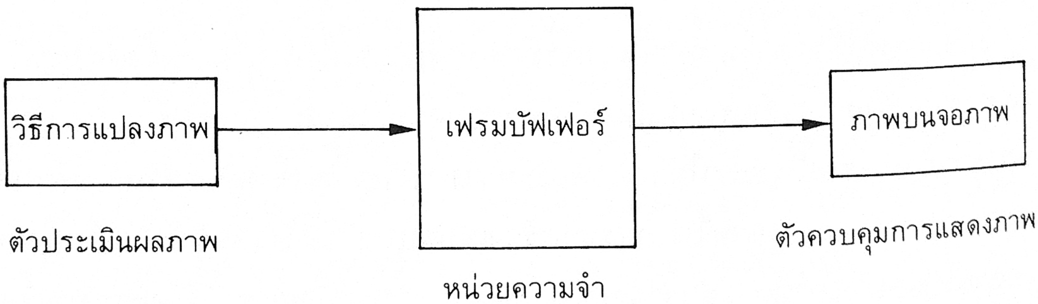
รูปที่ 1.4 รูปแบบการสร้างเส้นสแกน

ระบบจอภาพที่มีราคาไม่แพงนั้นมักจะมีอัตรารีเฟรชเท่ากับ 30 ครั้งต่อวินาที สำหรับรูปแบบการทำงานเพื่อสร้างเส้นสแกนนั้นจะไม่ทำตามลำดับเส้นที่ 1, เส้นที่ 2,... ดังคำอธิบายข้างต้น เพราะว่าถ้าทำเช่นนั้นจะทำให้จอภาพเกิดการกะพริบเนื่องจากอัตรารีเฟรชต่ำ เส้นสแกนช่วงบนของจอภาพกำลังจะจางหายไป การมาสร้างเส้นสแกนทับเส้นเดิมก็ช้าไปเล็กน้อยทำให้เรารู้สึกว่าภาพหายไปและจะทำให้เราเห็นว่าจอภาพกะพริบ การแก้ปัญหาทำได้โดยการเปลี่ยนรูปแบบการสร้างเส้นสแกนไปเล็กน้อย กล่าวคือ แทนที่จะสร้างทีละเส้นตามลำดับก็ให้แบ่งเป็น 2 ขั้นตอน ขั้นตอนแรกให้สร้างเส้นสแกนที่เป็นเลขคี่คือเส้นที่ 1, เส้นที่ 3, เส้นที่ 5, ... ก่อน แล้วค่อยมาสร้างเส้นสแกนที่เป็นเลขคู่ (แสดงดังรูปที่ 1.5) ผลก็คือ จะช่วยลดการกะพริบของจอภาพลงได้ เนื่องจากเวลาที่จะกลับมาสร้างเส้นสแกนทับเส้นเดิมที่เป็นเลขคี่นั้นเส้นสแกนที่เป็นเลขคู่ก็ยังคงอยู่ ทำให้เรารู้สึกว่าภาพยังไม่ได้จางหายไปนั่นคือไม่เกิดการกะพริบ สำหรับระบบจอภาพที่มีราคาแพงขึ้นคุณภาพดีขึ้น จะไม่ใช้วิธีนี้ แต่จะมีอัตรารีเฟรชสูงขึ้น เช่น 60 ครั้งต่อวินาที เป็นต้น



รูปที่ 1.5 รูปแบบการสร้างเส้นสแกนแบบ 2 ขั้นตอน

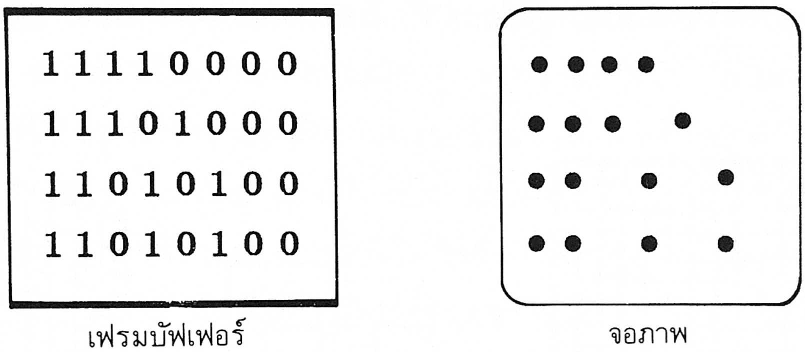
ในการแสดงภาพหนึ่งภาพจะต้องมีพิกเซลบางจุดที่ต้องสว่างและบางจุดต้องมืด การที่จะจัดการให้เกิดภาพตามที่ต้องการได้นั้นมีส่วนประกอบ 3 ส่วนที่จะใช้ในการจัดการนี้ คือ เฟรมบัฟเฟอร์ (frame buffer), ตัวควบคุมการแสดงภาพ (display controller) และวิธีการแปลงภาพให้เป็นตำแหน่งของพิกเซลที่เหมาะสมในเฟรมบัฟเฟอร์ (scan conversion algorithms) (ดูรูปที่ 1.6)



รูปที่ 1.6 ส่วนประกอบในหน่วยแสดงภาพแบบแรสเตอร์สแกน

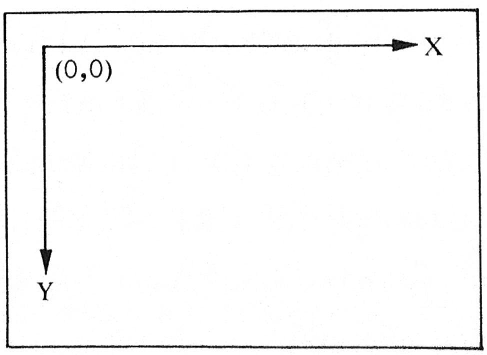
1. **เฟรมบัฟเฟอร์** พิกเซลหรือจุดแต่ละจุดที่ปรากฏอยู่บนจอภาพจะสอดคล้องกับค่าบิต (bit) ที่อยู่ในหน่วยความจำส่วนหนึ่ง ซึ่งเราจะเรียกหน่วยความจำส่วนนี้ว่าเฟรมบัฟเฟอร์ หรือ *บิตแมป (bit map)* บิตเหล่านี้จะถูกเก็บไว้ในลักษณะตาราง หน่วยความจำที่ใช้เป็นเฟรมบัฟเฟอร์ในระบบกราฟิกส์ปัจจุบันมักจะแยกออกจากหน่วยความจำหลักของเครื่องคอมพิวเตอร์เพื่อ

ที่จะทำให้สามารถแสดงภาพออกทางจอภาพได้อย่างรวดเร็วมากขึ้น จำนวนแถวของเฟรมบัฟเฟอร์จะเท่ากับจำนวนเส้นแรสเตอร์ที่จอภาพแสดงได้ และจำนวนหลักของเฟรมบัฟเฟอร์จะเท่ากับจำนวนพิกเซลที่มีได้ในเส้นแรสเตอร์แต่ละเส้น การบอกขนาดของหน่วยความจำที่ใช้เป็นเฟรมบัฟเฟอร์อาจบอกในรูปของจำนวนพิกเซลที่สามารถแสดงบนจอภาพหรืออาจจะบอกในรูปจำนวนพิกเซลในหลักคูณจำนวนพิกเซลในแถวก็ได้ ตัวอย่างเช่น 640 × 350 สำหรับจอภาพแบบ EGAHi และสำหรับจอภาพแบบ VGAHi จะเป็น 640 × 480 เป็นต้น เมื่อมีการใส่บิต 1 ลงในเฟรมบัฟเฟอร์ตรงตำแหน่งใดก็ตาม จะเกิดเป็นจุดสว่างบนจอภาพตรงตำแหน่งที่สอดคล้องกับเฟรมบัฟเฟอร์ เช่น ถ้าให้แถวที่ 2 หลักที่ 3 ของเฟรมบัฟเฟอร์มีค่าเป็นบิต 1 จอภาพที่ตำแหน่งแถวที่ 2 หลักที่ 3 ก็จะเกิดเป็นจุดสว่าง เป็นต้น ส่วนตำแหน่งที่มีค่าเป็นบิต 0 ก็จะไม่มีจุดสว่าง (ดูรูปที่ 1.7)



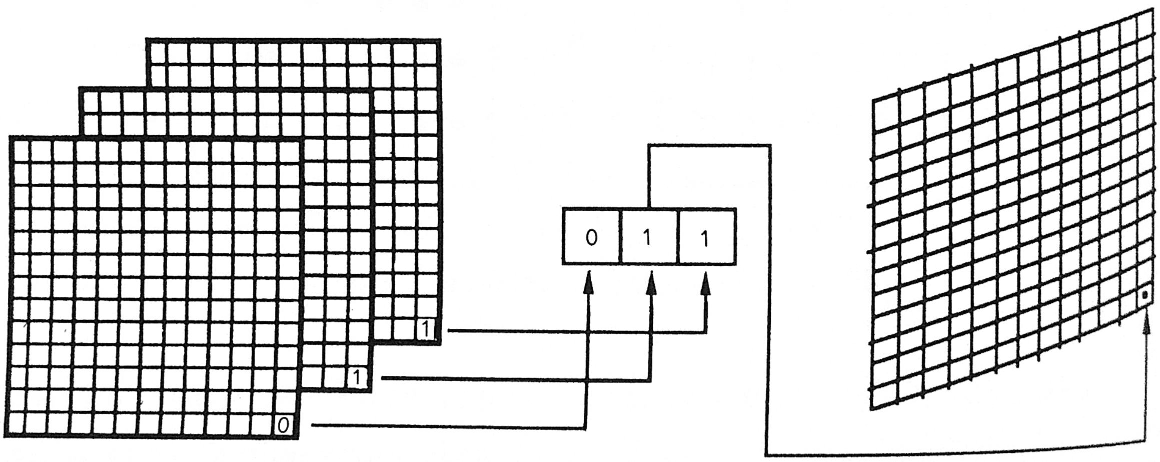
รูปที่ 1.7 ความสัมพันธ์ของเฟรมบัฟเฟอร์กับจอภาพ

แต่ละตำแหน่งพิกเซลบนจอภาพและตำแหน่งในหน่วยความจำที่สอดคล้องกันในเฟรมบัฟเฟอร์จะถูกอ้างถึงได้โดยใช้คู่ลำดับ (x,y) โดยที่ x จะแทนค่าตำแหน่งของหลัก ส่วน y แทนตำแหน่งของแถว จุด (0,0) ของระบบพิกัดนี้จะอยู่ที่มุมบนซ้ายของจอภาพดังรูปที่ 1.8 ระบบพิกัดของจอภาพ (screen coordinate) แบบนี้จะอธิบายในบทต่อไป



รูปที่ 1.8 ระบบพิกัดของจอภาพ

ข้อมูลในเฟรมบัฟเฟอร์ซึ่งใช้แทนพิกเซลแต่ละจุดนั้นจะประกอบด้วยบิตจำนวนหนึ่ง สำหรับจอภาพขาวดำซึ่งมีความเข้มเพียง 2 ระดับ ข้อมูลในเฟรมบัฟเฟอร์จะมีเพียง 1 บิต (1-bit-plane frame buffer) ก็พอ ซึ่งต่างกับจอภาพแบบสีหรือจอภาพแบบขาวดำที่มีความเข้มหลายระดับ ข้อมูลสำหรับ 1 พิกเซลจะต้องมีมากกว่า 1 บิต ในรูปที่ 1.9 แสดงเฟรมบัฟเฟอร์ที่ใช้ 3 บิต นั่นคือใน 1 พิกเซลจะมีค่าที่ใช้แทนพิกเซลนี้ได้ 8 ค่า (2^3) ซึ่งแต่ละค่าจะแทนความเข้ม 1 ระดับ รวมทั้งหมดก็แทนได้ 8 ระดับ จากระดับ 0 ถึงระดับ 2^3-1 = 7 สำหรับโทรทัศน์ขาวดำถ้าใช้ข้อมูล 8 บิตสำหรับ 1 พิกเซลก็จะสามารถแสดงระดับความเข้มได้ถึง 2^8 หรือ 256 ระดับและสำหรับระบบจอภาพสีต้องการข้อมูล 24 บิต (24- bit-plane frame buffer) โดยที่จะใช้ 8 บิตสำหรับแต่ละแม่สีคือ แดง เขียว และน้ำเงิน ซึ่งตามทฤษฎีจะสร้างสีได้ถึง 2^{24} ซึ่งเท่ากับ 16,777,216 สี สำหรับจอภาพที่มีความละเอียด 512 × 512 พร้อมทั้งมีสีได้ครบเต็มที่ จะต้องใช้หน่วยความจำถึง 512 × 512 × 24 = 6,291,456 บิต ซึ่งหน่วยความจำของไมโครคอมพิวเตอร์ราคาต่ำ ๆ ไม่สามารถมีหน่วยความจำขนาดนี้ได้ ดังนั้นข้อมูลต่อ 1 พิกเซลจึงมีแค่เพียง 1 ถึง 4 บิตเท่านั้น

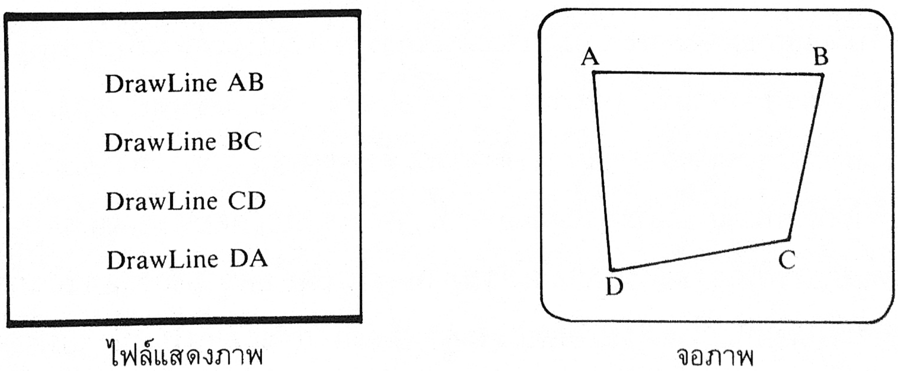


รูปที่ 1.9 เฟรมบัฟเฟอร์ที่ใช้ 3 บิตต่อพิกเซล

2. ตัวควบคุมการแสดงภาพ ส่วนที่ 2 ของหน่วยการแสดงภาพคือ ตัวควบคุมการแสดงภาพ ฮาร์ดแวร์ส่วนนี้จะอ่านค่าที่อยู่ในเฟรมบัฟเฟอร์ไปไว้ในวิดีโอบัฟเฟอร์ (video buffer) ซึ่งจะเปลี่ยนค่าบิตเหล่านี้ให้เป็นสัญญาณทางไฟฟ้าซึ่งใช้สำหรับควบคุมการแสดงภาพบนจอภาพ ตัวอย่างเช่น ถ้าตัวควบคุมการแสดงภาพพบค่าบิต 1 ในเฟรมนับเฟอร์ที่มีข้อมูล 1 บิตต่อพิกเซล ก็จะเกิดการส่งสัญญาณแรงดันสูงไปให้ CRT ซึ่งจะจัดการให้เกิดจุดสว่างบนจอภาพในตำแหน่งที่สอดคล้องกับข้อมูลที่อยู่ในเฟรมบัฟเฟอร์นั่นเอง สำหรับรายละเอียดการทำงานของตัวควบคุมการแสดงภาพจะไม่กล่าวถึงในที่นี้

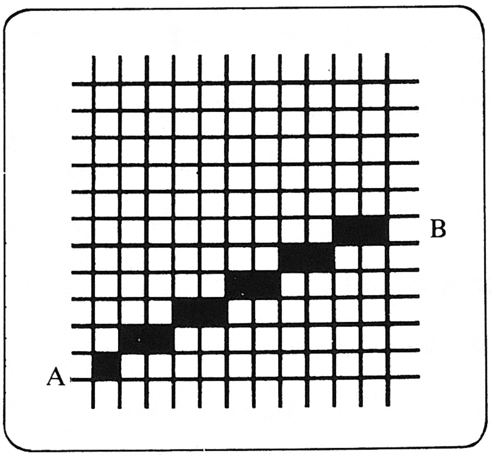
3. **วิธีการแปลงภาพให้เป็นตำแหน่งของพิกเซลที่เหมาะสมในเฟรมบัฟเฟอร์** ส่วนนี้เป็นวิธีการหรือกระบวนการที่ใช้ในการเปลี่ยนคำสั่ง หรือสมการทางกราฟิกส์ให้เป็นค่าที่เหมาะสม ซึ่งสามารถใช้แทนภาพที่ได้จากสมการหรือคำสั่งนั้นได้แล้วเก็บลงเฟรมบัฟเฟอร์ เช่น สมการเส้นตรง $x + y = 5$ หรือ คำสั่ง Line(a,b) จะต้องผ่านกระบวนการอันหนึ่งเพื่อให้ได้ค่าที่เหมาะสมแล้วเก็บในเฟรมบัฟเฟอร์ สำหรับระบบกราฟิกส์แบบแรสเตอร์สแกนที่มีคุณภาพสูงจะมีโปรเซสเซอร์จัดการการแสดงภาพโดยเฉพาะเพื่อทำกระบวนการข้างต้นรวมทั้งการจัดการอื่น ๆ เกี่ยวกับจอภาพ เช่น การลบจอภาพ เป็นต้น ส่วนระบบกราฟิกส์ที่เป็นไมโครคอมพิวเตอร์ราคาไม่แพงนัก จะใช้ CPU ของเครื่องกับโปรแกรมสำหรับจัดการงานดังกล่าวซึ่งทำให้การทำงานทำได้ช้ากว่ามาก ยากที่จะทำให้ระบบกราฟิกส์เป็นแบบอินเตอร์แอกตีฟได้ เนื่องจากในการเปลี่ยนแปลงภาพไปเพียงเล็กน้อยจะต้องมีการคำนวณมากมายตามมาเสมอ

(ข) จอภาพแบบแรนดอมเวกเตอร์ ระบบแรสเตอร์สแกนใช้เฟรมบัฟเฟอร์เพื่อช่วยในการแสดงภาพตามลำดับเส้นสแกนจากบนลงล่าง จากซ้ายไปขวา วิธีนี้มิใช่เป็นวิธีเดียวที่ใช้สร้างภาพได้ ยังมีอีกวิธีหนึ่งซึ่งเหมาะสำหรับการสร้างภาพที่เป็นเส้นอย่างเดียว ระบบนี้เรียกกันว่า *แรนดอมเวกเตอร์ (random vector)* หรือ *ระบบการประดิษฐ์ตัวอักษร (calligraphic systems)* หลักการของการสร้างภาพระบบนี้เป็นดังนี้ จะมีหน่วยความจำส่วนหนึ่งเรียกว่าไฟล์แสดงภาพ (display file) ซึ่งจะใช้สำหรับเก็บคำสั่งเกี่ยวกับการลากเส้น เช่น "ลากเส้นจากจุด A ไปจุด B" จากนั้น CPU จะจัดการกับคำสั่งเหล่านี้ให้ไปบังคับระบบเบี่ยงเบนประจุของ CRT เพื่อทำให้เกิดจุดสว่างเป็นเส้นตรงจากจุดที่กำหนดให้เป็นจุดเริ่มต้นไปยังจุดที่เป็นจุดปลาย (ดูรูปที่ 1.10) การแสดงภาพวิธีนี้ไม่จำเป็นต้องมีการรีเฟรชจอภาพทั้งจอเหมือนกับจอภาพแบบแรสเตอร์สแกน เพียงแค่รีเฟรชเฉพาะส่วนที่มีภาพซึ่งประกอบด้วยเส้นตรงก็พอแล้ว



รูปที่ 1.10 ไฟล์จอภาพกับหน้าจอของจอภาพที่สอดคล้องกัน

(ค) การเปรียบเทียบจอภาพแบบแรสเตอร์สแกนกับจอภาพแบบแรมดอมเวกเตอร์ จอภาพแบบแรนดอมเวกเตอร์สามารถวาดเส้นตรงได้ดีกว่าจอภาพแบบแรสเตอร์สแกนเนื่องจากว่าการแสดงภาพแบบแรสเตอร์สแกนจะแสดงได้เฉพาะตามเส้นสแกนเท่านั้น ดังนั้นสำหรับเส้นที่เป็นเส้นทแยงมุม เส้นที่วาดจะไม่เป็นเส้นตรงทีเดียวนักถ้าจำนวนของพิกเซลบนจอภาพไม่ละเอียดเพียงพอ (ดังรูปที่ 1.11) นอกจากนี้กระบวนการที่ใช้แปลงคำสั่งในไฟล์จอภาพให้เป็นจุดสว่างที่เหมาะสมบนจอภาพจะไม่ซับซ้อนเท่ากับกระบวนการที่ใช้ในแบบแรสเตอร์สแกน การเก็บข้อมูลของภาพแบบแรนดอมเวกเตอร์ต้องการเนื้อที่เก็บน้อยกว่าแบบเรสเตอร์สแกน เพราะว่าจะเก็บเพียงตำแหน่งจุดปลายของเส้นในภาพก็เพียงพอแล้ว ซึ่งต่างจากแบบแรสเตอร์สแกนที่ต้องเก็บข้อมูลจุดทั้งจอภาพ การทำงานกับภาพที่อยู่ในบางส่วนของจอภาพแบบแรนดอมเวกเตอร์จะทำได้เร็วกว่าแบบแรสเตอร์สแกน เพราะว่าการลบหรือการเปลี่ยนแปลงภาพทำได้โดยการลบหรือเปลี่ยนจุดพิกัดในไฟล์จอภาพเท่านั้น



รูปที่ 1.11 เส้นตรงบนจอภาพแบบแรสเตอร์สแกนกับปรากฏการณ์ขั้นบันได

สำหรับข้อเสียของจอภาพแบบแรนดอมเวกเตอร์ก็คือ มันไม่เหมาะกับภาพที่ต้องการระบายสีลงในพื้นที่และการสร้างภาพที่มีสีและแสงเงาเหมือนจริง ความซับซ้อนของภาพขึ้นกับความสามารถของระบบที่จะรีเฟรชเส้นของภาพได้แค่ไหน ตัวอย่างเช่น สามารถวาดเส้นได้ 10,000 เส้นในเวลา 1/60 วินาที เป็นต้น นั่นคือภาพหนึ่งภาพที่วาดขึ้นจะประกอบด้วยเส้นไม่เกิน 10,000 เส้น สำหรับจอภาพแบบแรสเตอร์สแกน การรีเฟรชภาพจะไม่ขึ้นกับความซับซ้อนของภาพ เพราะว่าข้อมูลไม่ว่ามากหรือน้อยที่อยู่ในเฟรมบัฟเฟอร์จะถูกแสดงออกที่จอภาพในหนึ่งรอบของการรีเฟรชเสมอ จากข้อดีของจอภาพแบบแรสเตอร์สแกนในการระบายสีและแสงเงาทำให้ระบบกราฟิกส์ในปัจจุบันส่วนใหญ่มักจะเลือกใช้จอภาพแบบแรสเตอร์สแกน

(ง) จอภาพแบบไม่ต้องการการรีเฟรช จอภาพแบบนี้เรียกกันว่า direct-view storage tube (DVST) ซึ่งเป็นจอภาพ CRT ที่ไม่ต้องการการรีเฟรช เนื่องจากจอภาพมีคุณสมบัติพิเศษสามารถทำให้จุดสว่างบนจอภาพคงอยู่ได้นานถึงประมาณ 1 ชั่วโมง ไม่จางหายไปแบบจอ CRT ทั่วไป ข้อดีคือเราจะได้ภาพที่ไม่เกิดการกะพริบเลย ข้อเสียคือ เราไม่สามารถลบภาพบางส่วนในจอภาพได้เพราะจอภาพแบบนี้การลบจะต้องลบทีเดียวทั้งจอภาพ

(จ) จอภาพแบบผลึกเหลว (liquid-crystal display,LCD) นี้เป็นจอภาพที่มีลักษณะพิเศษแตกต่างจากจอภาพแบบ CRT กล่าวคือ เป็นจอภาพที่ไม่ต้องมีหลอดภาพแบบจอภาพ CRT ทำให้จอภาพแบบนี้มีขนาดความหนาลดลง ข้อดีของจอภาพแบบนี้ก็คือ ใช้กำลังไฟฟ้าน้อย ขนาดและน้ำหนักเบาทำให้สามารถนำติดตัวไปไหนมาไหนได้สะดวก ในสมัยก่อนจอภาพแบบนี้จะมีสีได้เพียง 2 สีเท่านั้น แต่ในปัจจุบันสามารถให้สีมากกว่า 2 สีแล้ว แต่ว่าราคายังคงแพงมาก

(ฉ) อุปกรณ์รับข้อมูล ผู้ใช้ระบบอินเตอร์แอกตีฟกราฟิกส์สามารถติดต่อกับโปรแกรมกราฟิกส์ได้โดยอาศัยอุปกรณ์รับข้อมูล ในตอนนี้เราจะกล่าวถึงอุปกรณ์รับข้อมูลชนิดที่มีการใช้งานกันทั่วไปในระบบกราฟิกส์

อุปกรณ์รับข้อมูลที่เรารู้จักกันดีและมีใช้ในระบบกราฟิกส์เสมอก็คือ *แป้นพิมพ์ (keyboard)* เป็นอุปกรณ์ที่ใช้รับทั้งโปรแกรมและข้อมูลเข้าไปในคอมพิวเตอร์ เมื่อใดที่มีการกดแป้นพิมพ์รหัสที่ใช้แทนตัวอักษรนั้นก็จะถูกส่งเข้าไปในเครื่องคอมพิวเตอร์ และรหัสแต่ละตัวสามารถใช้แทนการทำงานอย่างหนึ่งในโปรแกรมกราฟิกส์ได้

อุปกรณ์รับข้อมูลอีกชนิดหนึ่งซึ่งปัจจุบันมีการใช้กันมากพอ ๆ กับการใช้แป้นพิมพ์ก็คือ *เมาส์ (mouse)* เมาส์เป็นอุปกรณ์สำหรับการกำหนดตำแหน่งภาพและเลือกภาพที่ต้องการซึ่งปรากฏอยู่บนจอภาพของคอมพิวเตอร์ได้อย่างสะดวกและแม่นยำ การทำงานของเมาส์จะเป็นดังบนจอภาพจะมีสัญลักษณ์อย่างใดอย่างหนึ่งสำหรับเป็นตัวชี้ เช่น รูปลูกศร หรือรูปเครื่องหมายบวก ตัวชี้นี้จะเคลื่อนที่ไปบนจอภาพได้ตามการเคลื่อนที่ของเมาส์ บนตัวของเมาส์จะมีปุ่มซึ่งอาจจะมีปุ่มเดียวหรือหลายปุ่มก็ได้ ถ้ามีการกดปุ่มบนเมาส์จะหมายถึงการเลือกภาพหรือบริเวณทำงานซึ่งตัวชี้กำลังชี้อยู่ การเคลื่อนที่และการกดปุ่มของเมาส์ก็คือการส่งข้อมูลเข้าเครื่องคอมพิวเตอร์นั่นเอง

ยังมีอุปกรณ์รับข้อมูลชนิดอื่น ๆ อีกซึ่งเราไม่ค่อยจะได้เห็นมากนัก แต่ก็ยังมีการใช้งานกันในระบบกราฟิกส์รุ่นเก่า ๆ หรือไม่ก็ใช้ในงานเฉพาะอย่าง เช่น *แพดเดิล (paddle), จอยสติ๊ก*

(joystick) ซึ่งใช้เป็นอุปกรณ์ในการเล่นเกมคอมพิวเตอร์, *ปากกาแสง (light pen)* เป็นอุปกรณ์รับข้อมูลสมัยแรก ๆ ของระบบกราฟิกส์, *ดิจิไตเซอร์ (digitizer)* เป็นอุปกรณ์ที่มีราคาค่อนข้างแพงจึงไม่นิยมนำมาใช้ในงานทั่วไป ดิจิไตเซอร์จะประกอบด้วยแผ่นรองสำหรับการวาด และอุปกรณ์ประเภทเมาส์ซึ่งอาจมีรูปร่างเหมือนเมาส์หรือปากกาก็ได้ แผ่นรองจะมีพิกัดคล้ายกับพิกัดบนจอภาพ ดังนั้นเมื่อเราใช้ปากกาของดิจิไตเซอร์ (stylus) วาดลงบนแผ่นรองก็จะเกิดเป็นภาพปรากฏบนจอภาพได้เหมือนกับภาพที่เราร่างบนแผ่นรอง งานที่เหมาะกับการใช้ดิจิไตเซอร์ได้แก่งานทำแผนที่คอมพิวเตอร์ (GIS) และงานทางด้าน CAD เป็นต้น

(ช) อุปกรณ์การแสดงผลบนกระดาษ ในระบบกราฟิกส์จะมีการใช้อุปกรณ์การแสดงผล 2 แบบคือ *เครื่องพิมพ์ (printer)* กับ *เครื่องวาด (plotter)* ซึ่งมีลักษณะการทำงานแตกต่างกัน

เครื่องพิมพ์ที่ใช้กันในปัจจุบันมีหลายชนิดได้แก่ *เครื่องพิมพ์แบบจุด (dot-matrix printer)* เป็นเครื่องพิมพ์ที่มีราคาถูกมาก ภาพที่ได้จากการพิมพ์จึงมีคุณภาพต่ำจนถึงปานกลาง ความละเอียดในการพิมพ์อยู่ในช่วง 10-120 จุดต่อนิ้ว ปัจจุบันคุณภาพในการพิมพ์ของเครื่องพิมพ์ชนิดนี้ได้สูงขึ้นมากกว่าเดิมแต่ราคาก็เพิ่มสูงมากขึ้นเช่นกัน เครื่องพิมพ์อีกชนิดหนึ่งซึ่งคุณภาพของภาพที่พิมพ์ได้มีคุณภาพดีคือ *เครื่องพิมพ์แบบฉีดหมึก (ink-jet printer)* เครื่องพิมพ์แบบนี้มีความสามารถในการพิมพ์ภาพสีและภาพที่มีแสงเงาได้เป็นอย่างดี ความละเอียดจะเป็น 200 จุดต่อนิ้วขึ้นไป นอกจากนี้ยังมี *เครื่องพิมพ์แบบเลเซอร์ (laser printer)* ซึ่งใช้หลักการเดียวกันกับเครื่องถ่ายเอกสารจะให้ภาพที่มีคุณภาพดีมาก ราคาของอุปกรณ์ชนิดนี้จึงมักจะสูงตามไปด้วย หลักการทำงานของเครื่องพิมพ์จะเหมาะกับการพิมพ์ภาพที่เป็นแบบแรสเตอร์สแกนมากกว่าแบบแรนดอมเวกเตอร์ เนื่องจากการพิมพ์จะพิมพ์ในลักษณะลอกภาพบนจอภาพลงบนกระดาษทีละเส้นสแกน ส่วนภาพที่เป็นแบบแรนดอมเวกเตอร์ อุปกรณ์การแสดงผลที่เหมาะสมก็คือเครื่องวาด

เครื่องวาดที่มีใช้ในระบบกราฟิกส์มีหลายชนิด การแบ่งชนิดจะแบ่งตามลักษณะการทำงาน เช่น *เครื่องวาดแบบระนาบ (flatbed plotter)* การบังคับปากกาของเครื่องวาดจะต้องใช้ 2 แกนคือ แกนกว้างและแกนยาว กระดาษจะวางบนแท่นแล้วลากปากกาไปบนกระดาษ อีกชนิดเรียกว่า *เครื่องวาดแบบทรงกระบอก (drum plotter)* กระดาษจะถูกม้วนอยู่ในแกนหมุนแล้วเคลื่อนที่เพื่อให้ปากกาลากเส้นได้ถูกตำแหน่ง ส่วนปากกาก็จะเคลื่อนที่ในแกนเดียวเท่านั้น

เครื่องวาดสามารถให้ผลภาพเส้นที่มีสีได้เป็นอย่างดีและมีคุณภาพของเส้นสูง แต่ภาพที่มีการระบายสีและแสงเงาจะทำได้ไม่ดีนัก เนื่องจากจะช้าและเปลืองหมึกพิมพ์มาก ดังนั้นอุปกรณ์แบบนี้จึงเหมาะกับการแสดงผลภาพที่เป็นแบบแรนดอมเวกเตอร์

1.5 ซอฟต์แวร์ในระบบกราฟิกส์

ซอฟต์แวร์กราฟิกส์เป็นส่วนสำคัญพื้นฐานที่ระบบการแสดงภาพกราฟิกส์ทุก ๆ ระบบจะต้องมี ในสมัยแรกเริ่มซอฟต์แวร์ทางกราฟิกส์จะมีราคาแพง แต่ปัจจุบันราคาของซอฟต์แวร์กราฟิกส์ถูกลงมากจนใคร ๆ ก็หามาใช้ได้

ในปี ค.ศ. 1979 คณะกรรมการวางแผนมาตรฐานซอฟต์แวร์ทางกราฟิกส์ (graphics standard planning committee, GSPC) ได้พยายามจัดการให้เกิดมาตรฐานของซอฟต์แวร์ทางกราฟิกส์ขึ้นในอเมริกา ซึ่งได้ระบบ CORE (core graphics system) ซึ่งเป็นมาตรฐานสำหรับระบบการแสดงภาพ 3 มิติ และในเวลาใกล้เคียงกัน สถาบันกำหนดมาตรฐานของประเทศเยอรมนี (West German National Standards, DIN) ก็ได้พัฒนามาตรฐานกราฟิกส์ของตัวเองขึ้นเรียกว่า GKS (graphical kernel system) ซึ่งเป็นมาตรฐานสำหรับระบบการแสดงภาพ 2 มิติเท่านั้นขาดส่วนที่เป็น 3 มิติ ในปี ค.ศ. 1982 ระบบ GKS ถูกใช้เป็นมาตรฐานนานาชาติของระบบกราฟิกส์ *สมาคมกราฟิกส์นานาชาติ (International Graphics Community)* ก็เลยพยายามรวมมาตรฐาน CORE กับ GKS เข้าเป็นมาตรฐานเดียวกัน แต่ไม่สำเร็จเนื่องจากมีการเมืองเข้ามาเกี่ยวข้อง

ต่อมาก็ได้มีการพัฒนามาตรฐานขึ้นอีกหลายมาตรฐาน ทั้งที่เกิดขึ้นใหม่ และที่ขยายจากระบบมาตรฐานเดิมตัวอย่างเช่น GKS-3D, PHIGS (Programmer's Hierarchical Interface Graphics System), PHIGS+ เป็นต้น

สำหรับหนังสือเล่มนี้ เราจะไม่ยึดมาตรฐานใดมาตรฐานหนึ่งโดยเฉพาะ แต่จะกล่าวถึงหลักการเบื้องต้นที่สำคัญในระบบกราฟิกส์ และจะกำหนดคำสั่งที่เกี่ยวกับกราฟิกส์เอง โดยที่เวลานำไปเขียนโปรแกรมตัวอย่างเราจะใช้วิธีเทียบกับคำสั่งที่มีในเทอร์โบซี ทั้งนี้เพื่อให้โปรแกรมที่เขียนขึ้นนั้นไม่ผูกติดกับเทอร์โบซีเพียงอย่างเดียว

1.6 สรุป

ในบทนี้เราได้กล่าวถึงเรื่องทั่ว ๆ ไปของระบบกราฟิกส์อันได้แก่ ประวัติย่อ ๆ ของการพัฒนาระบบกราฟิกส์ การประยุกต์ใช้กราฟิกส์ในงานด้านต่าง ๆ และได้อธิบายถึงฮาร์ดแวร์และซอฟต์แวร์ที่มีการใช้ในระบบกราฟิกส์ ปัจจุบันระบบกราฟิกส์เริ่มถูกนำมาใช้ในงานต่าง ๆ มากขึ้นทำให้ราคาของระบบซึ่งเคยสูงมากค่อย ๆ ลดลง และขนาดของระบบก็เล็กลงเรื่อย ๆ เช่นกันโดยที่คุณภาพและความสามารถสูงขึ้น

2 การสร้างภาพกราฟิกส์เบื้องต้น

2.1 บทนำ

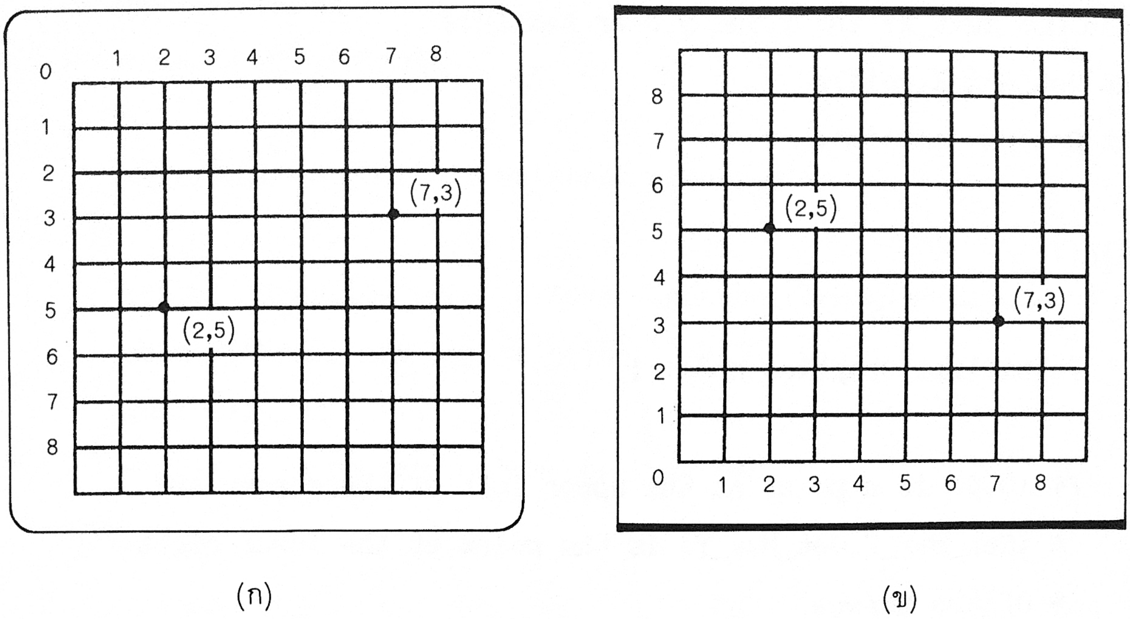
ภาพกราฟิกส์ที่คอมพิวเตอร์สร้างขึ้นนั้นจะสร้างขึ้นได้โดยใช้ภาพกราฟิกส์เบื้องต้นต่าง ๆ ซึ่งได้แก่ จุด (points), เส้นตรง (straight lines), เส้นโค้ง (curves), และภาพรูปทรงเรขาคณิตต่าง ๆ (geometric figures) เช่น วงกลม, วงรี หรือรูปเหลี่ยม เป็นต้น นอกจากนี้ยังต้องประกอบด้วยคำสั่งที่เกี่ยวกับการจัดการหน้าจอ เช่น การลบหน้าจอ, การวางภาพที่กำหนดไว้ในตำแหน่งที่ต้องการบนจอภาพ เป็นต้น

ในช่วงก่อนปี ค.ศ. 1980 ฮาร์ดแวร์ (hardware) และซอฟต์แวร์ (software) ของคอมพิวเตอร์สามารถช่วยนักเขียนโปรแกรมประยุกต์พัฒนาโปรแกรมสำเร็จรูปทางกราฟิกส์ได้โดยใช้วิธีทำโปรแกรมย่อยรวบรวมไว้ให้เลือกใช้ตามที่ต้องการ

สำหรับในบทนี้เราจะมาศึกษาถึงการสร้างภาพกราฟิกส์เบื้องต้น เพื่อเก็บรวบรวมไว้เป็นโปรแกรมย่อยสำหรับการสร้างภาพกราฟิกส์อื่น ๆ ต่อไป

2.2 การสร้างจุด

ภาพบนจอภาพแบบแรสเตอร์สแกนเกิดจากจุดสว่างหลาย ๆ จุดซึ่งจะกำหนดตำแหน่งได้โดยกำหนดจุดในเฟรมบัฟเฟอร์ที่สอดคล้องกับจุดจริงบนจอภาพ ทั้งจอภาพและเฟรมบัฟเฟอร์จะใช้ระบบจุดพิกัด 2 มิติในการอ้างถึงจุดต่าง ๆ โดยมีจุดกำเนิดหรือจุด (0,0) อยู่ที่มุมบนซ้ายของจอภาพ (รูปที่ 2.1(ก)) ซึ่งต่างจากระบบพิกัดที่เรามักใช้ในการเขียนกราฟ กล่าวคือ จุดกำเนิดอยู่ที่มุมล่างซ้าย (รูปที่ 2.1(ข)) การอ้างถึงพิกเซลใดพิกเซลหนึ่งจะใช้คู่ลำดับ (x,y) โดยที่ x และ y เป็นจำนวนเต็มบวกหรือศูนย์ ดังตัวอย่างในรูปที่ 2.1



รูปที่ 2.1 ระบบพิกัดของจอภาพและเฟรมบัฟเฟอร์เทียบกับระบบพิกัดที่เราใช้กัน

โปรแกรมที่ 2.1 เป็นโปรแกรมสำหรับปรับจอภาพเพื่อให้สามารถแสดงภาพกราฟิกส์ได้ โดยภาพกราฟิกส์ที่แสดงนั้นจะเป็นภาพระบบพิกัดที่ใช้ รวมทั้งข้อความอธิบาย ซึ่งในโปรแกรมนี้จะกำหนดให้คำสั่งสำหรับปรับจอภาพเป็น Initialize__Graphics__Mode ซึ่งในโปรแกรมจะไปเรียกใช้คำสั่ง initgraph ของเทอร์โบซีมาใช้นั่นเอง และคำสั่งสำหรับเลิกใช้จอภาพแบบกราฟิกส์จะเป็น Close__Graphics__Mode ซึ่งก็คือคำสั่ง closegraph ในเทอร์โบซี สำหรับภาพที่แสดงบนจอภาพก็จะต้องใช้คำสั่งกราฟิกส์อื่น ๆ ด้วย ซึ่งเราจะใช้คำสั่งเหล่านี้ไปก่อนแล้วค่อยอธิบายการทำงานของคำสั่งเหล่านี้ในภายหลัง

โปรแกรมที่ 2.1

```
/* A C program that demonstrates how to initialize graphic screen
 * and shows the picture of screen coordinate system with some text
*/

#include <graphics.h>


void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line(int x1,int y1,int x2,int y2);
```

```
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);

main()
{
    Initialize_Graphics_Mode();

    /* (0,0) is a point at the upper left of the screen and
     * (Get_Max_X,Get_Max_Y) is the point at the lower right
     * of the screen
    */

    Draw_Line(0,0,Get_Max_X(),0);
    Draw_Line(0,0,0,Get_Max_Y());
    Out_Text_XY(Get_Max_X()-15,5,"X");
    Out_Text_XY(5,Get_Max_Y()-15,"Y");
    Out_Text_XY(5,5,"ORIGIN");
    Out_Text_XY(Get_Max_X()/2-124,Get_Max_Y()/2-22,
      "COORDINATE SYSTEM OF THE SCREEN");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");
```

```
    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}

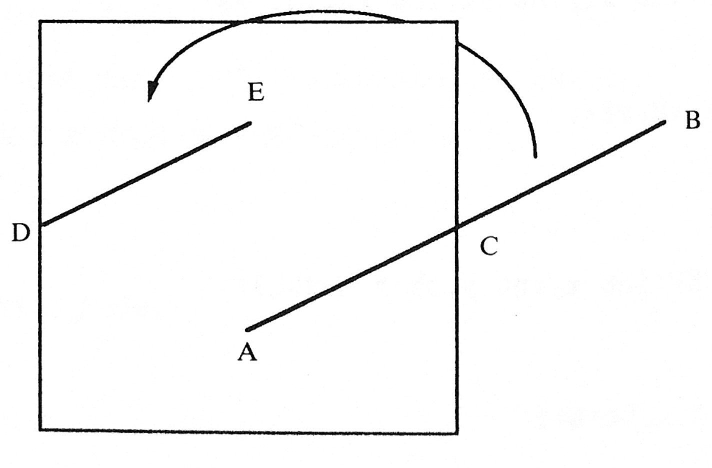
void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}
```

```
int Get_Max_Y(void)
{
    return(getmaxy());
}
```

ในการสร้างภาพกราฟิกส์ทุกครั้งเราจะต้องใช้คำสั่ง Initialize__Graphics__Mode เพื่อปรับจอภาพก่อนเสมอจึงจะทำให้จอภาพสามารถแสดงผลเป็นภาพได้ และควรจะต้องใช้คำสั่ง Close__Graphics__Mode เพื่อเลิกใช้จอภาพแบบกราฟิกส์ควบคู่ไปด้วย

ค่าของจุดพิกัด x และ y จะต้องมีค่าไม่เกินค่าขอบเขตของจอภาพที่ใช้ ถ้ามีการกำหนดค่าเกินขอบเขตจะต้องมีการจัดการอย่างใดอย่างหนึ่งเพื่อกันเหตุการณ์เช่นนี้ ตัวอย่างเช่น ให้จุดที่มีค่าอยู่เกินขอบเขตไม่ว่าจะเป็นขอบเขตบน ล่าง ซ้าย หรือขวา ถูกเปลี่ยนค่าให้เป็นค่าที่ขอบเขต นั่นคือภาพที่มีพิกัดเกินขอบเขตจะถูกตัดออกไปเลย (clipping) อีกวิธีเป็นการให้ค่าที่เกินขอบเขตไปเริ่มจากจุดเริ่มต้นอีกที (ดูรูปที่ 2.2) ดังนั้นจุดต่าง ๆ ของภาพส่วนที่เกินขอบเขตทางขวาไปจะมาปรากฏทางด้านซ้ายของจอภาพแทน ซึ่งเรียกว่า *ผลวนรอบ (wrap around effect)* สำหรับการอธิบายในตอนนี้จะถือว่าภาพอยู่ในขอบเขตของจอภาพทั้งหมด



รูปที่ 2.2 ปรากฏการณ์วนรอบ

ในการสร้างจุดให้ปรากฏบนจอภาพหนึ่งจุด เราจะกำหนดให้ใช้คำสั่งดังนี้

Plot__Point(x,y)

ซึ่งจะทำให้เกิดจุดสว่างที่ตำแหน่ง (x,y) บนจอภาพ สำหรับในเทอร์โบซีจะใช้คำสั่งว่า putpixel ซึ่งเราจะให้คำสั่ง Plot__Point เรียกใช้คำสั่ง putpixel ของเทอร์โบซีนั่นเอง ส่วนค่าของ x และ y เรามักจะกำหนดให้เป็นตัวเลขจำนวนเต็มที่ไม่ใช่จำนวนเต็มลบเสมอ

โปรแกรมที่ 2.2 แสดงตัวอย่างโปรแกรมการใช้คำสั่ง Plot__Point เพื่อสร้างจุดในตำแหน่งต่าง ๆ บนจอภาพ

```
/* A C program that demonstrates how to use Plot_Point command
 * and shows the points on the screen in specific positions and
 * colors
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Out_Text_XY(int x,int y,char text[]);

main()
{
    int x = 50,y = 50,color = 15;
    int i;

    Initialize_Graphics_Mode();

    Out_Text_XY(100,300,
        "Plot Points in specific positions and colors");
    Plot_Point(x,y,color);
    Out_Text_XY(x+5,y-5,"(50,50)");
    for(i=60;i<=200;i+=10)
      Plot_Point(i,i,color);
```

```
    getchar();
    Close_Graphics_Mode();
    return(0);
}
void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Plot_Point(int x,int y,int color)
{
    putpixel(x,y,color);
```

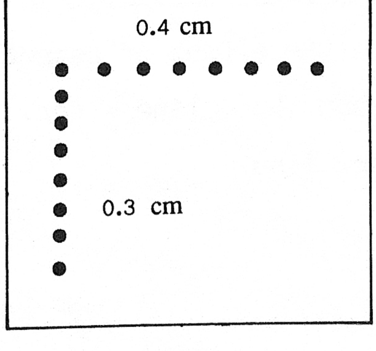
```
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}
```

จากโปรแกรม คำสั่ง Plot__Point ก็คือชื่อของฟังก์ชันฟังก์ชันหนึ่งซึ่งไปเรียกใช้คำสั่งการพล็อตจุด (putpixel) ของเทอร์โบซีนั่นเอง สมมติว่าเราไม่ได้ใช้เทอร์โบซีแต่เราใช้ตัวแปรภาษาซีตัวอื่นซึ่งคำสั่งที่ใช้ในการพล็อตจุดนั้นอาจจะแตกต่างจากเทอร์โบซี เช่น อาจจะเป็น point (x, y, color) เป็นต้น โปรแกรมของเราก็ยังสามารถใช้งานได้ โดยเราจะเปลี่ยนเพียง putpixel ซึ่งอยู่ในฟังก์ชัน Plot__Point ให้เป็น point เท่านั้นเอง

2.3 อัตราส่วนแอสเป็กต์

ถ้าเราใช้คำสั่งสร้างพิกเซลขึ้นมา 8 พิกเซลลงบนจอภาพทั้งในแนวดิ่งและแนวนอน แสดงดังรูปที่ 2.3 แล้วลองวัดความยาวของพิกเซลที่ต่อติดกันทั้ง 8 พิกเซลจะพบว่าความยาวในแนวนอนและในแนวดิ่งมีค่าต่างกันแม้ว่าจะมีจำนวนพิกเซลเท่ากัน จากรูปที่ 2.3 เราได้ว่าพิกเซล**ในแนวนอน**มีความยาว 0.4 เซนติเมตร ส่วนในแนวดิ่งยาว 0.3 เซนติเมตร อัตราส่วนระหว่างความ**ยาวในแนวนอน**กับความยาวในแนวดิ่ง = $\frac{0.4}{0.3}$ หรือประมาณ 1.33 นี้เรียกว่า ***อัตราส่วนแอสเป็กต์ (aspect ratio)*** ค่านี้เกิดจากการที่ระยะห่างระหว่างพิกเซลในแนวนอนของจอภาพมีค่าไม่เท่ากับระยะห่างระหว่างพิกเซลในแนวดิ่งของจอภาพนั่นเอง ซึ่งต่อมาในปัจจุบันมีการปรับปรุงให้ระยะห่างระหว่างพิกเซลนี้มีขนาดเท่ากันทั้งในแนวนอนและในแนวดิ่งจึงทำให้อัตราส่วนแอสเป็กต์มีค่าเป็น 1



รูปที่ 2.3 อัตราส่วนแอสเป็กต์ซึ่งมีค่าเป็น 1.33

อัตราส่วนนี้มีผลต่อความถูกต้องของภาพที่วาดบนจอภาพ ตัวอย่างเช่น เราต้องการวาดรูปสี่เหลี่ยมจัตุรัสซึ่งมีความกว้างเท่ากับ 80 พิกเซล สำหรับจอภาพซึ่งมีอัตราส่วนแอสเป็กต์เป็น $\frac{4}{3}$ ความยาวจริงในแนวนอนบนจอภาพจะเท่ากับ 0.4 × 10 = 4 เซนติเมตร (8 พิกเซล = 0.4 เซนติเมตร) และถ้าเราลากเส้นในแนวดิ่งโดยใช้จำนวนพิกเซลเท่าเดิมคือ 80 พิกเซล ความยาวจริงของเส้นบนจอภาพจะสั้นกว่าเส้นที่อยู่ในแนวนอน นั่นคือจะยาวเพียง 0.3 × 10 = 3 เซนติเมตร ทำให้ภาพที่ได้ไม่ใช่สี่เหลี่ยมจัตุรัสตามที่ต้องการ การแก้ไขทำได้โดยใช้จำนวนพิกเซลมากกว่า 80 พิกเซล ซึ่งจะทำให้ความยาวจริงของเส้นในแนวดิ่งบนจอภาพเท่ากับ 4 เซนติเมตรด้วย จำนวนพิกเซลที่ต้องการหาได้ $80\times\frac{4}{3}$ ซึ่งมีค่าประมาณ 107 พิกเซล หมายความว่าในการวาดรูปสี่เหลี่ยมจัตุรัสบนจอภาพที่มีอัตราส่วนแอสเป็กต์เป็น $\frac{4}{3}$ เราต้องใช้พิกเซล 80 พิกเซล สำหรับการวาดเส้นในแนวนอนและใช้พิกเซล 107 พิกเซล สำหรับการวาดในแนวดิ่ง เราจึงจะได้รูปสี่เหลี่ยมจัตุรัสตามต้องการ

การวาดรูปวงกลมก็มีปัญหาเช่นเดียวกันกับการวาดรูปสี่เหลี่ยมจัตุรัส กล่าวคือ ถ้าไม่ได้คำนึงถึงอัตราส่วนแอสเป็กต์ ภาพวงกลมที่ปรากฏบนจอภาพก็จะเป็นรูปวงรี การแก้ไขก็ทำได้เช่นเดียวกับการวาดรูปสี่เหลี่ยมจัตุรัส นั่นคือ ความยาวในแนวดิ่งจะต้องคูณด้วยค่าอัตราส่วนแอสเป็กต์ก่อนแล้วจึงนำไปวาดลงบนจอภาพ สำหรับการพิมพ์ภาพลงบนกระดาษก็ต้องคำนึงถึงอัตราส่วนแอสเป็กต์ของเครื่องพิมพ์ด้วย ภาพที่ได้จึงมีอัตราส่วนที่ถูกต้อง

โปรแกรมที่ 2.3 แสดงผลของอัตราส่วนแอสเป็กต์ที่มีต่อภาพเส้นในแนวดิ่งและในแนวนอน ถ้าค่าอัตราส่วนแอสเป็กต์ของจอภาพเป็น 1 ความยาวของเส้นในแนวนอนและในแนวดิ่งที่ปรากฏบนจอภาพจะเท่ากัน แต่ถ้าค่าอัตราส่วนแอสเป็กต์ไม่เท่ากับ 1 เส้นในแนวนอนและในแนวดิ่งจะมีความยาวไม่เท่ากัน เนื่องจากมีการวาดเส้นตรงจึงกำหนดให้คำสั่งสำหรับวาดเส้นตรงคือ Draw_Line ซึ่งจะไปเรียกใช้คำสั่ง line ในเทอร์โบซี

โปรแกรมที่ 2.3

```
/* A C program that shows the effect of aspect ratio in drawing lines,
 * if aspect ratio = 1 then (hor. line = ver. line)
 * else (hor. line <> ver. line)
*/


#include <graphics.h>
```

```
void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line(int x1,int y1,int x2,int y2);
void Out_Text_XY(int x,int y,char text[]);

main()
{
    int x = 100, y = 100;

    Initialize_Graphics_Mode();

    /* draw a horizontal and a vertical line */
    Draw_Line(0,0,x,0);
    Out_Text_XY(x+10,10,
          "Horizontal line");
    Draw_Line(0,0,0,y);
    Out_Text_XY(10,y,
          "Vertical line");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{

    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");
```

```
    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

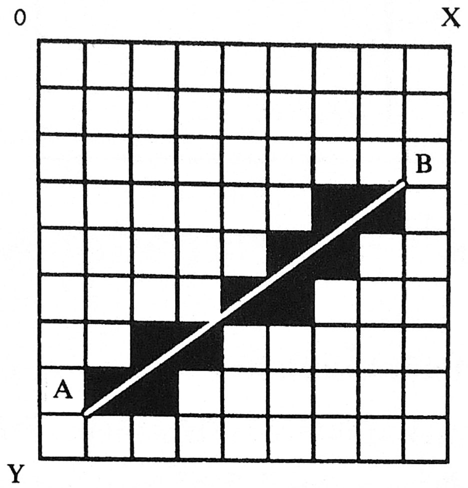
void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}
```

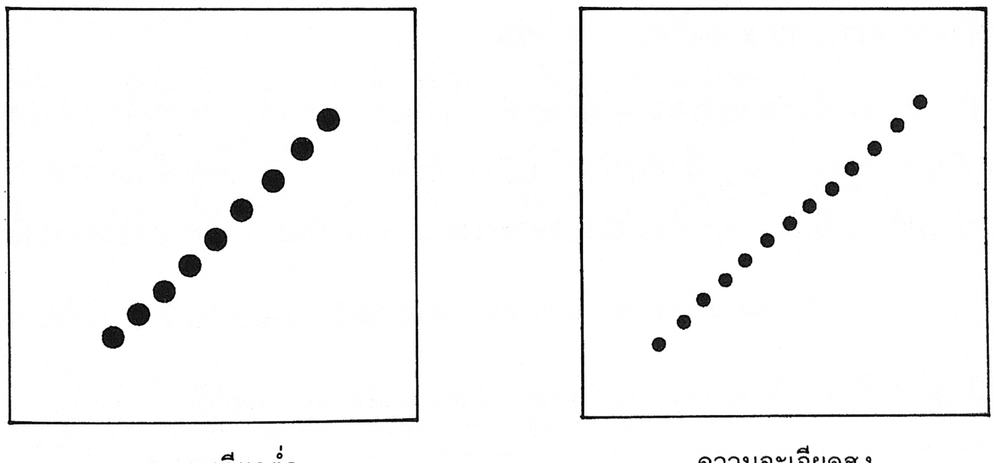
2.4 การวาดเส้นตรง

เส้นตรงก็คือพิกเซลที่จัดเรียงเป็นลำดับติด ๆ กันในแนวตรง สำหรับจอภาพแบบแรสเตอร์สแกน การลากเส้นตรงในแนวเฉียง เราจำเป็นต้องเลือกพิกเซลที่ใกล้กับแนวเส้นที่สุดเพื่อให้ได้เส้นตรงที่ดีที่สุด รูปที่ 2.4 แสดงเส้นที่ลากบนจอภาพแบบแรสเตอร์สแกน ความถูกต้องและคุณภาพของ

เส้นที่แสดงบนจอภาพจะขึ้นอยู่กับความละเอียดของจอภาพ ถ้าเป็นจอภาพที่มีความละเอียดสูง เช่น 1024 × 1024 จุด จะสามารถวาดเส้นได้ตรงและต่อเนื่องมากกว่าจอภาพที่มีความละเอียดต่ำ เส้นที่ปรากฏบนจอภาพที่มีความละเอียดต่ำจะมีช่องว่างระหว่างพิกเซลให้เห็นทำให้เส้นไม่ต่อเนื่อง รูปที่ 2.5 แสดงภาพเส้นตรงที่ปรากฏบนจอภาพที่มีความละเอียดต่างกัน

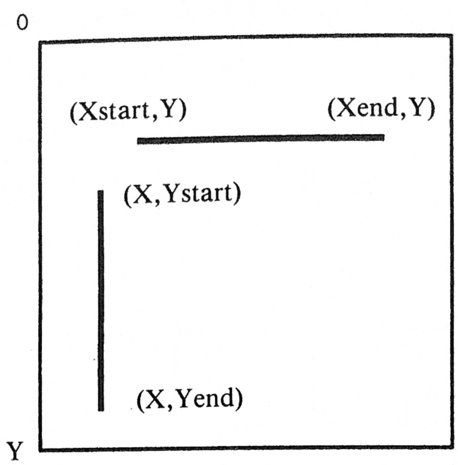


รูปที่ 2.4 พิกเซลที่ใช้สำหรับประกอบเป็นเส้นตรง AB



รูปที่ 2.5 เปรียบเทียบภาพเส้นตรงที่ได้จากจอภาพที่มีความละเอียดต่างกัน

ในระบบกราฟิกส์ที่มีความสามารถสูง การวาดเส้นตรงจะทำโดยทางฮาร์ดแวร์ซึ่งจะทำให้สามารถวาดเส้นได้อย่างรวดเร็วมาก ส่วนระบบกราฟิกส์ที่มีความสามารถต่ำ ก็จะวาดเส้นโดยใช้ซอฟต์แวร์ซึ่งวาดได้ช้ากว่ามาก ในการวาดเส้นเรามักจะต้องเป็นคนกำหนดจุดเริ่มต้นกับจุดสิ้นสุดเอง แล้วระบบกราฟิกส์จะวาดเส้นเชื่อมจุดที่เรากำหนดไว้ ในตอนต่อไปนี้เราจะมากล่าวถึงวิธีต่าง ๆ ที่ใช้ในการวาดเส้นตรงเชื่อมจุด 2 จุดอย่างคร่าว ๆ



รูปที่ 2.6 เส้นแนวนอนและเส้นแนวดิ่ง

2.4.1 เส้นแนวนอนและเส้นแนวดิ่ง

เส้นแนวนอนและเส้นแนวดิ่งเป็นเส้นที่วาดได้ง่ายที่สุด ถ้าค่าของ x ที่จุดเริ่มต้นน้อยกว่าค่าของ x ที่จุดสิ้นสุด การวาดเส้นแนวนอนทำได้โดยให้ค่าทางแกน y คงที่ แล้วเพิ่มค่าทางแกน x ขึ้นทีละ 1 พิกเซล ดังแสดงในรูปที่ 2.6 เป็นภาพเส้นแนวนอนซึ่งลากจากจุด (Xstart,Y) ไปยัง (Xend,Y) และ Xstart < = Xend แต่ถ้า Xstart > Xend เราก็จะทำกลับกัน กล่าวคือ ให้ค่า y คงที่ แล้วลดค่าทางแกน x ลงทีละ 1 พิกเซล

ส่วนเส้นแนวดิ่งก็สามารถวาดได้โดยให้ค่าทางแกน x คงที่ แล้วเพิ่มค่าทางแกน y ขึ้นทีละ 1 พิกเซล ถ้าลากเส้นจากจุด y ซึ่งมีค่าน้อยไปยัง y ที่มีค่ามากกว่า แต่ถ้าต้องการลากเส้นจากจุด y ที่มีค่ามากไปยัง y ที่มีค่าน้อย ก็จะต้องลดค่าทางแกน y ลงทีละ 1 พิกเซลในขณะที่ค่าทางแกน x คงที่

โปรแกรมที่ 2.4 โปรแกรมการลากเส้นแนวนอนและเส้นแนวดิ่งโดยใช้คำสั่ง Plot_Point

```
/* A C program that shows how to draw horizontal and vertical lines
 * using the command Plot_Point
*/


#include <graphics.h>


void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
```

```
void Plot_Point(int x,int y, int color);
void Out_Text_XY(int x,int y,char text[]);

main()
{
    int x,y;

    Initialize_Graphics_Mode();

    /* draw a horizontal line when xstart <= xend */
    y = 50;
    for(x = 0 /* xstart */;x <= 200 /* xend */;x++)
        Plot_Point(x,y,15);
    Out_Text_XY(0,y+10,"Horizontal line : xstart <= xend");

    /* draw a horizontal line when xstart >= xend */
    y = 100;
    for(x = 200 /* xstart */;x >= 0 /* xend */;x--)
        Plot_Point(x,y,15);
    Out_Text_XY(0,y+10,"Horizontal line : xstart >= xend");

    /* draw a vertical line when ystart <= yend */
    x = 50;
    for(y = 150 /* ystart */;y <= 300 /* yend */;y++)
        Plot_Point(x,y,15);
    Out_Text_XY(5,310,"Vertical line : ystart <= yend");

    /* draw a vertical line when ystart >= yend */
    x = 300;
    for(y = 300 /* ystart */;y >= 150 /* yend */;y--)
        Plot_Point(x,y,15);
    Out_Text_XY(x,310,"Vertical line : ystart >= yend");
```

```
    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}
void Close_Graphics_Mode(void)
{
    closegraph();
}

void Plot_Point(int x,int y, int color)
{
    putpixel(x,y,color);
```

```
}


void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}
```

2.4.2 เส้นทแยงมุม

การวาดเส้นทแยงมุมซึ่งมีความลาดชันเป็น +1 เราสามารถทำได้โดยการเพิ่มค่าทางแกน x และ y ขึ้นทีละ 1 พิกเซลจากจุดเริ่มต้นซึ่งมีพิกัดน้อยกว่าพิกัดของจุดสิ้นสุด หรือการลดค่าทางแกน x และ y ลงทีละ 1 พิกเซลในกรณีกลับกัน สำหรับเส้นทแยงมุมที่มีความลาดชันเป็น -1 สามารถทำได้โดยการเพิ่มค่าทางแกน x และลดค่าทางแกน y ลงทีละ 1 พิกเซลพร้อม ๆ กัน

โปรแกรมที่ 2.5 โปรแกรมแสดงเส้นทแยงมุมที่มีความลาดชัน +1 และ -1 โดยใช้คำสั่ง Plot__Point

```
/* A C program that shows how to draw diagonal lines
 * using the command Plot_Point
*/


#include <graphics.h>


void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y, int color);
void Out_Text_XY(int x,int y,char text[]);
void Draw_Line(int x1,int y1,int x2,int y2);
int Get_Max_X(void);
int Get_Max_Y(void);


main()
{
```

```
int xstart,ystart,xend,yend,x,y;
int i = 0;

Initialize_Graphics_Mode();

Draw_Line(0,0,Get_Max_X(),0);
Draw_Line(0,0,0,Get_Max_Y());
Out_Text_XY(Get_Max_X()-15,5,"X");
Out_Text_XY(5,Get_Max_Y()-15,"Y");
Out_Text_XY(5,5,"ORIGIN");

/* draw a diagonal line with slope +1
   (using screen coordinate system, origin at top-left) */
xstart = 0; ystart = 0;
xend = 300; yend = 300;
/* initial point */
x = xstart; y = ystart;
/* to final point */
while (x+i <= xend) {
        Plot_Point(x+i,y+i,15);
        i++;
}
Out_Text_XY(100,20,"Diagonal line with slope = +1");

/* draw a diagonal line with slope -1
   (using screen coordinate system, origin at top-left) */
xstart = 0; ystart = Get_Max_Y();
xend = 300; yend = Get_Max_Y()-300;
/* initial point */
x = xstart; y = ystart; i = 0;
/* to final point */
while (x+i <= xend) {
```

```
            Plot_Point(x+i,y-i,15);
            i++;
        }
        Out_Text_XY(100,Get_Max_Y()-20,
            "Diagonal line with slope = -1");

        getchar();
        Close_Graphics_Mode();
        return(0);
}

void Initialize_Graphics_Mode(void)
{
        /* request auto detection */
        int gdriver = DETECT, gmode, errorcode;

        /* initialize graphics mode */
        initgraph(&gdriver, &gmode, "");
        /* check if error */
        errorcode = graphresult();

        if (errorcode != grOk) /* an error occurred */
        {
            printf("Graphics error : %s\n", grapherrormsg(errorcode));
            printf("Press any key to halt : ");
            getch();
            exit(1); /* return with error code */
        }
}

void Close_Graphics_Mode(void)
{
```

```
    closegraph();
}

void Plot_Point(int x,int y, int color)
{
    putpixel(x,y,color);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}
int Get_Max_Y(void)
{
    return(getmaxy());
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}
```

2.4.3 เส้นตรงที่มีความลาดชันใด ๆ

การวาดเส้นตรงที่มีความลาดชันเท่าไรก็ได้นั้น มีปัญหาในการสร้างหลายอย่าง ตัวอย่างเช่น จอภาพแบบแรสเตอร์สแกนสามารถแสดงจุดได้เฉพาะตรงที่เป็นตำแหน่งพิกเซลเท่านั้น ทำให้เส้นตรงที่ไม่ใช่เส้นแนวนอนหรือเส้นแนวดิ่งมีลักษณะเป็นขั้นบันได (staircase effect) ซึ่งเส้นตรง

เหล่านี้จะแทนเส้นตรงจริง ๆ ได้โดยประมาณเท่านั้น เพราะว่าพิกเซลที่มีอาจไม่อยู่ในตำแหน่งที่ตรงกับจุดบนส่วนของเส้นตรงที่จะสร้างจริง ๆ ดังนั้นเราจึงต้องเลือกจุดพิกเซลที่ใกล้กับจุดจริง ๆ มากที่สุดปัญหาอีกประการหนึ่งก็คือ การหาพิกเซลที่ใกล้หรือดีที่สุดนั้นสามารถทำได้หลายวิธี การเลือกวิธีใดในการสร้างเส้นประมาณนี้ก็ขึ้นกับความเร็วในการสร้างเส้นและภาพเส้นที่วาดได้ดูเป็นเส้นตรงมากแค่ไหน เพื่อให้เข้าใจเกณฑ์ในการเลือกวิธีสร้างเส้นตรง ในตอนนี้จะกล่าวถึงวิธีต่าง ๆ ที่ใช้ในการสร้างเส้นตรง

1. วิธีสร้างโดยใช้สมการเส้นตรง วิธีนี้จะใช้สมการเส้นตรง :

$$Y = (m \times X) + b$$

โดยที่ m คือ ความลาดชัน

b คือ ค่าของจุดที่เส้นตรงตัดแกน Y (จุดที่ค่า X เป็นศูนย์)

วิธีวาดเริ่มจากเราต้องหาค่าของ m และ b ก่อน จากนั้นเราจะวาดเส้นได้โดยการเพิ่มค่าของ X ขึ้นทีละหนึ่งหน่วยจากจุดเริ่มต้น (Xstart,Ystart) จนถึงจุดสิ้นสุด (Xend,Yend) และทุก ๆ ขั้นของการเพิ่มค่า X เราก็จะคำนวณหาค่าของ Y จากสมการ ก็จะได้จุดพิกัด (X,Y) ซึ่งเราจะทำให้พิกเซลที่พิกัดนี้สว่างโดยใช้คำสั่ง Plot_Point ค่าของ Y ที่ได้จากการคำนวณมักจะเป็นค่าจำนวนจริง เพราะฉะนั้นจึงต้องมีการปรับให้ค่าจำนวนจริงนี้เป็นค่าจำนวนเต็มก่อน เพื่อใช้เป็นค่าพิกัดได้

วิธีนี้มีปัญหา 2 ประการคือ ประการแรก ใช้เวลาในการวาดเส้นมาก และประการที่สองคือ เส้นซึ่งมีความลาดชันมากกว่า 1 จะวาดได้ไม่ดีเนื่องจากมีช่วงว่างระหว่างเส้นมากเพราะเป็นช่วงที่ค่า X จากจุดเริ่มไปยังจุดสิ้นสุดซึ่งใช้ในการคำนวณมีจำนวนน้อยเกินไป

2. วิธี Simple DDA (simple digital differential analyzer) เป็นวิธีที่ง่ายต่อการเขียนโปรแกรมและสามารถให้เส้นที่ค่อนข้างดี

สมมติว่าเราต้องการจะสร้างเส้นตรงซึ่งลากจากจุดพิกัด (Xstart,Ystart) ไปยังจุดพิกัด (Xend,Yend) ซึ่งมีความลาดชัน m ดังนี้

$$m = \frac{(Yend - Ystart)}{(Xend - Xstart)}$$

พิกัด (X_1,Y_1) และ (X_2,Y_2) ซึ่งเป็นจุดที่อยู่ติดกันและอยู่บนเส้นตรงนี้ อาจนำมาใช้ในการคำนวณหาค่าความลาดชัน m ได้ดังนี้

$$m = \frac{(Y_2 - Y_1)}{(X_2 - X_1)} \quad \text{.......... 2.1}$$

สำหรับค่าสมบูรณ์ของ m น้อยกว่า 1 (abs(m) < 1) และ Xstart < Xend เราสร้างเส้นโดยการเพิ่มค่าของ X ตัวก่อน (X_1) ขึ้น 1 หน่วยจนถึง Xend และคำนวณหาค่าของ Y (ถ้าในกรณีที่ Xstart > Xend, ให้สลับค่าจุดปลายทั้งสอง) ดังนั้นสำหรับจุดที่ตามมา (X_2)

$$X_2 = X_1 + 1$$

หรือ

$$X_2 - X_1 = 1$$

แทนค่า $X_2 - X_1 = 1$ ลงในสมการที่ 2.1 จะได้

$$Y_2 = Y_1 + m \quad \text{.......... 2.2}$$

จากสมการที่ 2.2 เราสามารถหาค่า Y ตัวต่อมา (Y_2) ได้จากค่า Y ตัวก่อน (Y_1) วิธีการหาค่าจุดต่อมาจากค่าจุดเดิมเช่นนี้เป็นเทคนิคที่ทำให้สามารถวาดเส้นได้เร็วขึ้น

ถ้า abs(m) > 1 และ Ystart < Yend เราจะสร้างเส้นได้โดยเพิ่มค่าของ Y จนถึง Yend และคำนวณหาค่าของ X (ถ้าในกรณีที่ Ystart > Yend, ให้สลับค่าจุดปลายทั้งสอง) ดังนั้น

$$Y_2 = Y_1 + 1$$

หรือ

$$Y_2 - Y_1 = 1$$

แทนค่า $Y_2 - Y_1 = 1$ ลงในสมการที่ 2.1 จะได้

$$X_2 = X_1 + 1/m \quad \text{.......... 2.3}$$

จากสมการที่ 2.3 แสดงให้เห็นว่าเราสามารถหาค่า X ตัวต่อมาได้จากค่า X ตัวเก่า

จากการคำนวณค่า X หรือ Y ข้างต้น ค่าที่ได้จะเป็นค่าจำนวนจริง ดังนั้นก่อนที่จะนำไปใช้ในการกำหนดจุดพิกเซลจะต้องแปลงให้เป็นค่าจำนวนเต็มก่อน โปรแกรมต่อไปนี้เป็นตัวอย่างโปรแกรมสำหรับวาดเส้นตรงโดยใช้วิธี Simple DDA ในโปรแกรมจะมีการทดสอบว่า $X_1 = X_2$

หรือไม่ด้วย เนื่องจากถ้า $X_1 = X_2$ จะเกิดการหารด้วยศูนย์ขึ้นซึ่งจะทำให้เกิดข้อผิดพลาดได้ และมีการทดสอบว่า $Y_1 = Y_2$ หรือไม่ด้วย เพื่อกันไม่ให้ใช้โปรแกรมนี้วาดเส้นแนวนอน (การวาดเส้นแนวนอนไม่จำเป็นต้องใช้โปรแกรมยาว ๆ เช่นนี้) ซึ่งโปรแกรมจะไม่ทำงานเมื่อพบสองเงื่อนไขนี้เท่านั้น

โปรแกรมที่ 2.6 การสร้างเส้นตรงโดยใช้วิธี Simple DDA โดยใช้คำสั่ง Plot__Point

```
/* A C program that demonstrates how to draw a line
 * using Simple DDA method
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Swap(int x1,int y1,int x2,int y2);
void Draw_DDA_Line(int x1,int y1,int x2,int y2);

main()
{

    Initialize_Graphics_Mode();

    Draw_DDA_Line(100,50,200,200);
    Out_Text_XY(100,Get_Max_Y()-100,
         "Lines drawing using Simple DDA algorithm");

    getchar();
    Close_Graphics_Mode();
```

```
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_DDA_Line(int x1,int y1,int x2,int y2)
/* (x1,y1) => start point, (x2,y2) => end point */
{
    int xi,yi; /* incremental values */
    int xr,yr; /* computed values */
    float m,mi; /* slope and inverse slope */
```

```
    if ((x1 == x2) || (y1 == y2))
          Out_Text_XY(100,Get_Max_Y()-200,
          "Out of scope of this program");
    else {
         m = (y2-y1)/(x2-x1); /* slope */
         if (((m<1) && (x1>x2)) || ((m>1) && (y1>y2)))
             /* check if start values are less than end values */
             Swap(x1,y1,x2,y2); /* swap endpoints */
         Plot_Point(x1,y1,15); /* plot first point */
         if (m < 1) { /* increment x values by 1 */
               yr = y1;
               for(xi=x1+1; xi<=x2-1; xi++){/* calculate y and plot*/
                    yr = yr + m;
                    Plot_Point(xi,yr,15);
               }
    }
else { /* slope > 1 */
    mi = 1/m; xr = x1;
    for(yi=y1+1; yi<=y2-1; yi++) { /* calculate y and plot */
          xr = xr + mi;
          Plot_Point(xr,yi,15);
    }
    }
}
} /* Draw_DDA_Line */

void Plot_Point(int x,int y,int color)
{
    putpixel(x,y,color);
}

void Out_Text_XY(int x,int y,char text[])
```

```
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}

void Swap(int x1,int y1,int x2,int y2)
{
    int temp;

    temp = x2; x2 = x1; x1 = temp;
    temp = y2; y2 = y1; y1 = temp;
}
```

โปรแกรมข้างต้นนี้ยังไม่สมบูรณ์เพราะไม่สามารถวาดเส้นได้ในทุกกรณี แต่ในที่นี้จะไม่กล่าวถึงรายละเอียดในทุกกรณี เนื่องจากปัจจุบันนี้ภาษาโปรแกรมส่วนใหญ่จะเตรียมคำสั่งที่ใช้ในการวาดเส้นให้แล้ว ไม่จำเป็นต้องสร้างเอง ดังนั้นเราเพียงแค่เรียนรู้ว่ามีวิธีการอย่างไรบ้างก็พอ รูปที่ 2.7 แสดงการใช้วิธี Simple DDA เพื่อสร้างเส้นตรงจากจุดพิกัด (2,2) ถึง (6,7)

Xstart	=	2	Xend	=	6
Ystart	=	2	Yend	=	7
m	=	$\frac{(7 - 2)}{(6 - 2)}$	minverse	=	$\frac{4}{5}$
	=	$\frac{5}{4}$		=	0.8
	=	1.25			

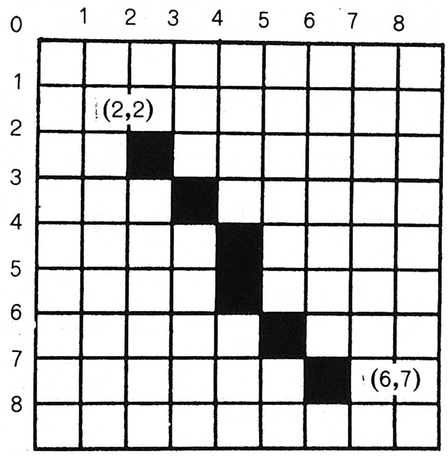
Iteration

1. Yr = 3
   Xr = 2 + 0.8 = 2.8
   Plot_Point(3,3)

2. Yr = 4
   Xr = 2.8 + 0.8 = 3.6
   Plot_Point(4,4)

3. Yr = 5
   Xr = 3.6 + 0.8 = 4.4
   Plot_Point(4,5)

4. Yr = 6
   Xr = 4.4 + 0.8 = 5.2
   Plot_Point(5,6)



รูปที่ 2.7 Simple DDA

3. วิธี Integer DDA เนื่องจากวิธี Simple DDA มีการคำนวณที่ใช้เวลามากคือการบวกเลขทศนิยม ดังนั้นจึงมีการพัฒนาวิธีที่ใช้เวลาน้อยกว่าโดยใช้การคำนวณที่เป็นจำนวนเต็มอย่างเดียวเท่านั้น ซึ่งเรียกว่าเป็นวิธี Integer DDA

หลักการเบื้องต้นของวิธีนี้ก็คือ แทนค่าความลาดชันซึ่งเป็นจำนวนจริงด้วยจำนวนเต็ม นั่นคือจะนำค่าของ delta_y = Yend-Ystart และ delta_x = Xend-Xstart มาใช้โดยตรง แทนที่จะนำไปคิดความลาดชันซึ่งจะต้องมีการคำนวณเลขทศนิยม

วิธีนี้แบ่งออกเป็น 4 กรณีขึ้นอยู่กับค่าของความลาดชันของเส้น แต่ละกรณีก็จะมีกฎเกณฑ์ของตัวมันเอง ซึ่งจะได้อธิบายแต่ละกรณีไป สำหรับค่า error ที่ใช้ในการอธิบายจะหมายถึงระยะทางระหว่างจุดที่พล็อตเพื่อสร้างเส้นกับจุดสิ้นสุดของเส้นที่ควรจะเป็นจริง ๆ และจุดสิ้นสุดจะต้องอยู่ในลำดับที่เหมาะสมของแต่ละกรณี

กรณีที่ 1 *ค่าความลาดชันระหว่าง 0 ถึง 1 โดยที่ delta__x และ delta__y เป็นค่าบวก :*

ถ้าค่าของ error เป็นลบ ให้พล็อตจุดต่อไปทางขวาของจุดเดิมที่พึ่งจะพล็อตมาโดยเลื่อนไปหนึ่งหน่วย แล้วบวกค่า delta__y เข้ากับค่า error แต่ถ้าค่าของ error เป็นบวก ให้พล็อตจุดต่อไปที่ตำแหน่งสูงขึ้นไปหนึ่งหน่วยและไปทางขวาอีกหนึ่งหน่วยจากจุดเดิม แล้วบวกค่า error ด้วยค่าของ delta__y - delta__x

---

กรณีที่ 2 *ค่าความลาดชันมากกว่า 1 โดยที่ delta__x และ delta__y เป็นค่าบวก :*

ถ้าค่าของ error เป็นลบ ให้พล็อตจุดต่อไปที่ตำแหน่งสูงขึ้นไปหนึ่งหน่วยและไปทางขวาอีกหนึ่งหน่วยจากจุดเดิม แล้วบวกค่า error ด้วยค่าของ delta__y - delta__x แต่ถ้าค่าของ error เป็นบวก ให้พล็อตจุดต่อไปที่ตำแหน่งสูงขึ้นไปหนึ่งหน่วยจากจุดเดิมที่พึ่งจะพล็อตมา แล้วลบค่า delta__x ออกจากค่า error

---

กรณีที่ 3 *ค่าความลาดชันระหว่าง -1 ถึง 0 โดยที่ delta__y เป็นบวก delta__x เป็นลบ :*

ถ้าค่าของ error เป็นลบ ให้พล็อตจุดต่อไปทางซ้ายของจุดเดิมที่พึ่งจะพล็อตมาโดยเลื่อนไปหนึ่งหน่วย แล้วบวกค่า delta__y เข้ากับค่า error แต่ถ้าค่าของ error เป็นบวก ให้พล็อตจุดต่อไปที่ตำแหน่งสูงขึ้นไปหนึ่งหน่วยและไปทางซ้ายอีกหนึ่งหน่วยจากจุดเดิม แล้วบวกค่า error ด้วยค่าของ delta__x + delta__y

---

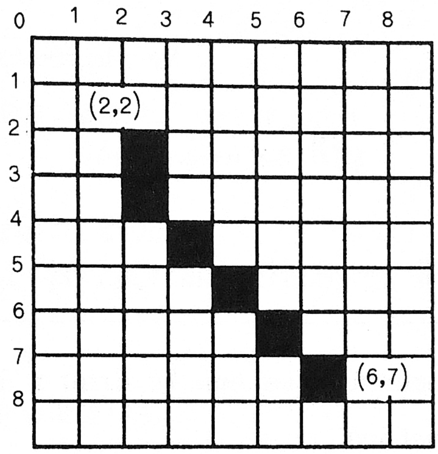
กรณีที่ 4 *ค่าความลาดชันน้อยกว่า -1 โดยที่ delta__y เป็นบวก delta__x เป็นลบ :*

ถ้าค่าของ error เป็นลบ ให้พล็อตจุดต่อไปที่ตำแหน่งสูงขึ้นไปหนึ่งหน่วยและไปทางซ้ายอีกหนึ่งหน่วยจากจุดเดิม แล้วบวกค่า error ด้วยค่าของ delta__x + delta__y แต่ถ้าค่าของ error เป็นบวก ให้พล็อตจุดต่อไปที่ตำแหน่งสูงขึ้นไปหนึ่งหน่วยจากจุดเดิมที่พึ่งจะพล็อตมาแล้วบวกค่า delta__x เข้ากับค่า error

---

ค่า error ที่จุดเริ่มต้นและจุดสิ้นสุดของเส้นจะมีค่าเป็น 0 และจะมีค่าเป็น 0 ทุกครั้งที่ตำแหน่งจุดที่คำนวณได้ อยู่บนเส้นที่ควรจะเป็นจริงและแต่ละกรณีข้างต้นจะมีค่าของ error ต่างกัน

รูปที่ 2.8 แสดงภาพของเส้นตรงที่ควรจะเป็นเทียบกับเส้นที่เกิดจากการสร้างโดยใช้วิธี Integer DDA เป็นเส้นที่ลากระหว่างจุด (2,2) ไปยังจุด (6,7) ในรูปแสดงการคำนวณจุดที่จะต้องพล็อตต่อไปจากจุดตั้งต้นจนถึงจุดสิ้นสุดด้วย จะเห็นว่าวิธีนี้ไม่จำเป็นต้องคำนวณค่าความลาดชัน (ซึ่งใช้การหาร) เลย แต่เราก็สามารถสร้างเส้นตรงได้



Xstart = 2 Xend = 6

Ystart = 2 Yend = 7

delta__x = 4 delta__y = 5

delta__y − delta__x = 1

การคำนวณในตัวอย่างนี้เป็น กรณีที่ 2 เพราะว่าความลาดชันมีค่ามากกว่า 1 ซึ่งดูได้จาก ค่าของ delta__y - delta__x = 1

Iteration :

1. error = 0
   x = 2
   y = 2 + 1 = 3
   error = 0 - 4 = -4

2. error = -4
   x = 2 + 1 = 3
   y = 3 + 1 = 4
   error = -4 + 1 = -3

3. error = -3
   x = 3 + 1 = 4
   y = 4 + 1 = 5
   error = -3 + 1 = -2

4. error = -2
   x = 4 + 1 = 5
   y = 5 + 1 = 6
   error = -2 + 1 = -1

5. error = -1
   x = 5 + 1 = 6
   y = 6 + 1 = 7
   error = -1 + 1 = 0

รูปที่ 2.8 Integer DDA

โปรแกรมที่ 2.7 การสร้างเส้นตรงโดยใช้วิธี Integer DDA

```
/* A C program that demonstrates how to draw a line
 * using Integer DDA method
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Out_Text_XY(int x,int y,char text[]);
void Swap(int x1,int y1,int x2,int y2);
void Draw_Integer_DDA_Line(int x1,int y1,int x2,int y2);

main()
{
    Initialize_Graphics_Mode();

    Draw_Integer_DDA_Line(50,50,300,200);
    Out_Text_XY(100,300,
```

```
        "Lines drawing using Integer DDA algorithm");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");
    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Integer_DDA_Line(int x1,int y1,int x2,int y2)
```

```
/* (x1,y1) => start point, (x2,y2) => end point */
{
    int dx,dy, /* rates of change */
    count, /* number of points */
    error, /* distance between endpoint and plotted points */
    x,y;    /* plotted points */

    /* initialize */
    error = 0;
    dx = x2 - x1;
    dy = y2 - y1;
    if (dy < 0 ) {  /* check if dy positive swap endpoints */
        Swap(x1,y1,x2,y2);
        dy = -dy; /* change dy values */
        dx = -dx;
    }
    Plot_Point(x1,y1,15); /* plot first starting point */
    x = x1; y = y1; /* initialize x and y values */

    if (dx >= 0) {  /* case 1 or case 2 */
       if (dx >= dy) { /* case 1 */
          for(count=1; count<=dx-1; count++) {
          /* plot up to the next last points */
       if (error < 0) {
          x++;
          Plot_Point(x,y,15);
          error += dy;
       }
       else { /* error >= 0 */
          x++; y++;
          Plot_Point(x,y,15);
          error += dy-dx;
```

```
            }
        }
    }
    else {  /* case 2 */
        for(count=1; count<=dy-1; count++) {
            if (error < 0) {
                x++; y++;
                Plot_Point(x,y,15);
                error += dy-dx;
            }
            else {  /* error >= 0 */
                y++;
                Plot_Point(x,y,15);
                error -= dx;
            }
        }
    }
}
else { /* dx < 0  case 3 or case 4 */
        if (dx >= dy) { /* case 3 */
            for(count=1; count<=dx-1; count++) {
                if (error < 0) {
                    x--;
                    Plot_Point(x,y,15);
                    error += dy;
                }
                else { /* error >= 0 */
                    x--; y++;
                    Plot_Point(x,y,15);
                    error += dx+dy;
                }
        }
```

```
 }
 else { /* case 4 */
     for(count=1; count<=dy-1; count++) {
          if (error < 0) {
               x--; y++;
               Plot_Point(x,y,15);
               error += dx+dy;
          }
          else { /* error >= 0 */
               y++;
               Plot_Point(x,y,15);
               error += dx;
          }
     }
}
}

     Plot_Point(x2,y2,15); /* plot last point *
 } /* Draw_Integer_DDA_Line */

void Plot_Point(int x,int y,int color)
{
     putpixel(x,y,color);
}

void Out_Text_XY(int x,int y,char text[])
{
     outtextxy(x,y,text);
}

void Swap(int x1,int y1,int x2,int y2)
{
     int temp;
```

```
    temp = x2; x2 = x1; x1 = temp;
    temp = y2; y2 = y1; y1 = temp;
}
```

2.4.4 คำสั่งสำหรับการสร้างเส้น

ระบบกราฟิกส์โดยทั่วไปนั้นจะมีคำสั่งสำหรับสร้างเส้นตรงเตรียมไว้ให้แล้วเสมอ ผู้ใช้เพียงแต่กำหนดจุดเริ่มต้นและจุดสิ้นสุดเท่านั้น หรือบางครั้งก็กำหนดเพียงจุดสิ้นสุดอย่างเดียว สำหรับคำสั่งที่เราใช้ในการเขียนโปรแกรมตัวอย่างในหนังสือเล่มนี้จะอ้างอิงกับคำสั่งที่มีใช้ในเทอร์โบซี ซึ่งมีอยู่ด้วยกัน 3 คำสั่งดังนี้

Draw__Line ใช้สำหรับสร้างเส้นจากจุด (x1,y1) ไปยัง (x2,y2) รูปแบบของคำสั่งจะเป็นดังนี้

```
Draw__Line(x1,y1,x2,y2)
```

โดยที่ x1,y1,x2,y2 ต้องเป็นเลขจำนวนเต็ม

Draw__Line__Rel ใช้สำหรับสร้างเส้นจากจุด CP (current pointer) ซึ่งเป็นจุดที่แสดงตำแหน่งปัจจุบันของตัวชี้บนจอภาพไปยังจุดที่กำหนดซึ่งตำแหน่งของจุดนี้ได้มาจากค่า x และค่า y ของ CP บวกกับระยะทางซึ่งกำหนดในคำสั่ง รูปแบบของคำสั่งเป็นดังนี้

```
Draw__Line__Rel(dx,dy)
```

โดยที่ dx,dy คือค่าระยะทางตามแนวแกน x และ y และต้องเป็นเลขจำนวนเต็ม

Draw__Line__To ใช้สำหรับสร้างเส้นตรงจากจุด CP ไปยังจุด (x,y) ที่กำหนดในคำสั่ง รูปแบบของคำสั่งเป็นดังนี้

```
Draw__Line__To(x,y)
```

โดยที่ x,y เป็นจำนวนเต็ม

ในบางครั้งเราต้องการจะย้ายจุด CP ไปยังตำแหน่งต่าง ๆ บนจอภาพโดยไม่มีการลากเส้น คำสั่งที่ใช้ได้มีอยู่ 2 คำสั่งคือ

Move__To ใช้สำหรับเลื่อนจุด CP จากจุดปัจจุบันไปยังจุดที่กำหนด รูปแบบของคำสั่งคือ

```
Move__To(x,y)
```

โดยที่ (x,y) คือจุดที่ต้องการให้เป็นจุด CP

Move__Rel ใช้สำหรับการเลื่อนจุด CP ไปยังจุดใหม่โดยกำหนดระยะห่างระหว่างจุดเดิมกับจุดใหม่ให้ รูปแบบคำสั่งเป็นดังนี้

```
Move__Rel(dx,dy)
```

โดยที่ dx และ dy คือค่าระยะทางตามแนวแกน x และ y และต้องเป็นเลขจำนวนเต็ม

โปรแกรมที่ 2.8 การสร้างเส้นตรงโดยใช้คำสั่ง Draw__Line, Draw__Line__Rel, Draw__Line__To และการใช้คำสั่ง Move__To, Move__Rel ในการเลื่อนจุด CP

```
/* A C program that demonstrates how to use command
 * Draw_Line, Draw_Line_Rel, and Draw_Line_To
 * Move_To and Move_Rel
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
```

```
void Draw_Line(int x1,int y1,int x2,int y2);
void Draw_Line_Rel(int dx,int dy);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Move_Rel(int dx,int dy);

main()
{
    Initialize_Graphics_Mode();

    Draw_Line(-50,100,50,250); /* -50 is out of screen boundary */
    getchar();
    /* CP is at (0,0) (default) */
    Draw_Line_Rel(100,50);
    /* draw a line from point (50,250) along the x-axis 100 pixels
       and along the y-axis 50 pixels */
    /* CP is at (100,50) */
    getchar();
    Draw_Line_To(50,50); /* draw a line from CP to (50,50) */
    /* CP is at (50,50) */
    getchar();
    Move_To(100,100); /* CP is at (100,100) */
    getchar();
    Draw_Line_To(200,200);
    /* CP is at (200,200) */
    getchar();
    Move_Rel(-50,-50); /* change CP to (200-50,200-50) = (150,150) */
    getchar();
    Draw_Line_To(100,0);

    getchar();
    Close_Graphics_Mode();
    return(0);
```

```
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{

    closegraph();
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}
```

```
void Draw_Line_Rel(int dx,int dy)
{
    linerel(dx,dy);
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Move_Rel(int dx,int dy)
{
    moverel(dx,dy);
}
```

2.5 การวาดรูปวงกลม

วงกลมเป็นภาพกราฟิกส์พื้นฐานที่ใช้ประกอบในภาพต่าง ๆ ทั่วไป ในตอนนี้เราจะกล่าวถึงวิธีที่ใช้ในการสร้างภาพวงกลม

2.5.1 การใช้สมการวงกลมโดยตรง

สมการวงกลมเขียนได้ดังนี้

$$(x - Xc)^2 + (y - Yc)^2 = r^2 \qquad \text{.......... 2.4}$$

ถ้าจุดศูนย์กลางของวงกลมอยู่ที่จุด (0,0) สมการวงกลมจะลดรูปลงได้เป็น

$$x^2 + y^2 = r^2 \qquad \text{.......... 2.5}$$

แก้สมการที่ 2.4 หาค่า y ได้ดังนี้

$$\left.\begin{aligned} y &= Yc + \mathrm{sqrt}(r^2 - (x - Xc)^2) \\ \text{และ} \quad y &= Yc - \mathrm{sqrt}(r^2 - (x - Xc)^2) \end{aligned}\right\} \quad \ldots\ldots\ldots 2.6$$

การวาดรูปวงกลมทำได้โดย เพิ่มค่า x ขึ้นทีละ 1 หน่วย จาก -r จนถึง +r และใช้สมการที่ 2.6 หาค่า y ทั้งสองทุกขั้นตอน และนำค่า y ที่ได้ไปแปลงเป็นพิกัดเพื่อพล็อตบนจอภาพสร้างรูปวงกลมได้ตามต้องการ

วิธีการข้างต้นเป็นวิธีที่ใช้วาดรูปวงกลมอย่างไม่มีประสิทธิภาพ เนื่องจากเหตุผลหลายประการ ประการแรก เราไม่ได้ใช้ประโยชน์จากความสมมาตรของวงกลม นั่นคือถ้าเราคำนวณได้ค่าพิกัดจุดหนึ่งของวงกลม เราก็สามารถทราบค่าพิกัดของวงกลมอีก 7 จุดได้ทันที ประการที่สองเกี่ยวกับจำนวนเวลาที่ใช้ในการคำนวณ จะเห็นว่าวิธีข้างต้นมีการคำนวณหาค่ายกกำลังสองและการคำนวณหาค่ารากที่สองซ้ำ ๆ กันทุกครั้งที่หาค่าของ y ทำให้ใช้เวลาในการคำนวณมาก จากเหตุผลที่กล่าวมา ทำให้ไม่ใช้วิธีนี้ในการสร้างรูปวงกลม

2.5.2 การใช้สมการเชิงมุมของวงกลม

สมการวงกลมอาจจะเขียนในรูปสมการเชิงมุมได้ดังนี้

$$\left.\begin{aligned} x &= Xc + r\cos(\theta) \\ y &= Yc + r\sin(\theta) \end{aligned}\right\} \quad \ldots\ldots\ldots 2.7$$

โดยที่ θ มีค่าอยู่ระหว่าง 0 ถึง 2π เนื่องจาก ความยาวของส่วนโค้งมีค่าเท่ากับ $r \times \theta$ (มาจากสูตร เส้นรอบวงวงกลม = 2π r) และ r ซึ่งเป็นรัศมีของวงกลม มีค่าคงที่ ดังนั้นส่วนโค้งของวงกลมจึงขึ้นกับ θ จุดพิกัดที่พล็อตเพื่อลากเส้นต่อกันเป็นส่วนโค้งของวงกลมจึงขึ้นกับ θ ด้วย เมื่อเราเปลี่ยนค่าของ θ จาก 0 ถึง 2π โดยเปลี่ยนไปทีละ $d\theta$ แล้วพล็อตจุดพิกัดที่สอดคล้องกัน ผลสุดท้ายก็จะได้รูปวงกลมตามต้องการ

ในการใช้สมการวงกลมเชิงมุมหาจุดพิกัดที่เหมาะสมนั้น (สมการที่ 2.7) การคำนวณโดยใช้ฟังก์ชัน cos และ sin ทุกครั้งที่คำนวณหาพิกัดจะเสียเวลามาก ทำให้การพล็อตสร้างรูปวงกลมช้าลงไปได้ เพื่อปรับปรุงความเร็วในการพล็อต เราจะใช้เทคนิคการคำนวณจุดพิกัดใหม่จากพิกัด

เดิมซึ่งเคยใช้ในวิธีการสร้างเส้นตรงมาแล้ว มาใช้ในการสร้างรูปวงกลมด้วย โดยที่เราจะต้องทราบความสัมพันธ์ระหว่างพิกัดใหม่กับพิกัดเดิมดังนี้

สมมติว่าจุด (x1,y1) กับ (x2,y2) บนเส้นรอบวงของวงกลม เป็นจุดที่ติดกัน

$$\left.\begin{aligned} x1 &= r\cos(\theta) \\ y1 &= r\sin(\theta) \\ x2 &= r\cos(\theta + d\theta) \\ y2 &= r\sin(\theta + d\theta) \end{aligned}\right\} \quad \text{..........} \; 2.8$$

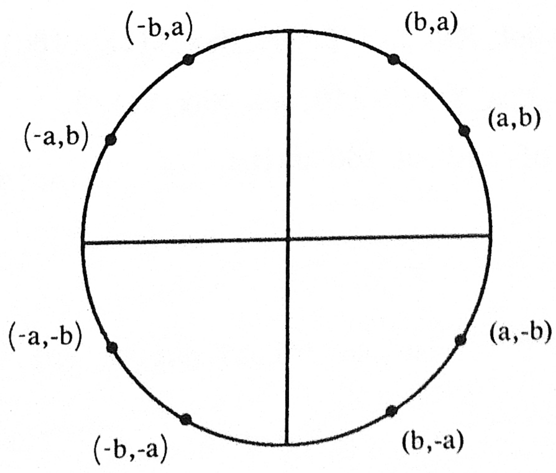
โดยที่ $d\theta$ เป็นค่ามุมที่เพิ่มขึ้น

จาก 4 สมการของสมการที่ 2.8 ข้างต้น เราสามารถเขียนสมการความสัมพันธ์ระหว่างพิกัด (x1,y1) และ (x2,y2) ได้ดังนี้

$$\left.\begin{aligned} x2 &= x1\cos(d\theta) - y1\sin(d\theta) \\ y2 &= y1\cos(d\theta) + x1\sin(d\theta) \end{aligned}\right\} \quad \text{..........} \; 2.9$$

นั่นคือเราสามารถหาค่าของพิกัด (x2,y2) ได้จากพิกัด (x1,y1) และมีการคำนวณหาค่าโดยใช้ฟังก์ชัน cos และ sin เพียงครั้งเดียวในตอนเริ่มต้นเท่านั้น

รูปที่ 2.9 แสดงจุดที่สมมาตรกันของวงกลม นั่นคือถ้าเราทราบค่าพิกัดจุดใดจุดหนึ่งใน 8 จุดนี้ เราก็จะสามารถพล็อตจุดที่พิกัดอีก 7 จุดได้ทันที



รูปที่ 2.9 จุดที่สมมาตรกัน 8 จุดของวงกลม

โปรแกรมที่ 2.9 การสร้างรูปวงกลมโดยใช้สมการวงกลมเชิงมุมและใช้เทคนิคการหาจุดพิกัดใหม่จากจุดพิกัดเดิม รวมทั้งใช้ความสมมาตรของวงกลมด้วย

```
/* A C program that demonstrates how to draw a circle using
 * incremental drawing, assume that aspect ratio = 1
*/

#include <graphics.h>
#include <math.h>
#define PI 3.14159

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Draw_Circle_1 (int xc,int yc,int radius);

main()
{
    Initialize_Graphics_Mode();

    Draw_Circle_1 (Get_Max_X()/2,Get_Max_Y()/2,150);
    Out_Text_XY(Get_Max_X()/2-110,Get_Max_Y()/2,
          "A circle of radius 150 units ");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
```

```
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Plot_Point(int x,int y,int color)
{
    putpixel(x,y,color);
}

void Draw_Circle_1 (int xc,int yc,int radius)
{
     double dtheta,   /* angle increment */
            cosval,sinval, /* values of cos(dtheta) and sin(dtheta) */
```

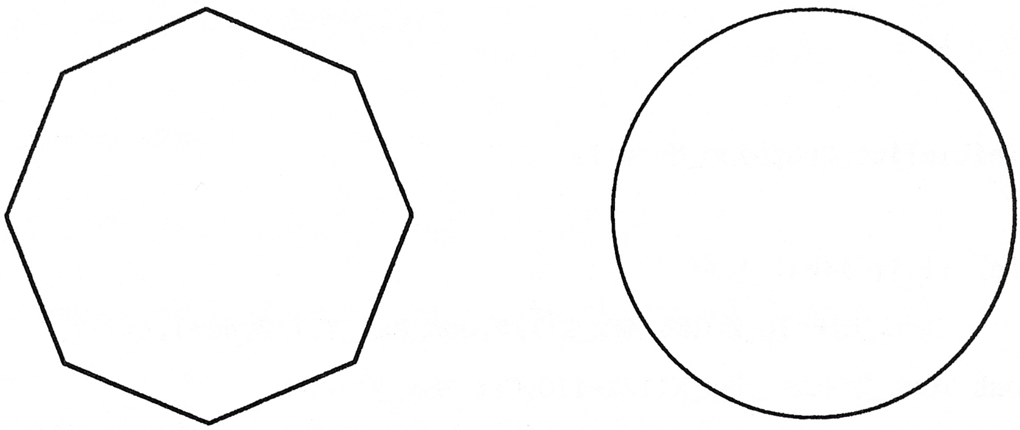
```
    x,y,   /* computed point */
    xtemp;  /* old value of x */
    /* initialize */
    dtheta = 2*PI/radius;
    cosval = cos(dtheta); sinval = sin(dtheta);
    /* initial coordinates */
    x = 0; y = radius;
    while (y >= x) {  /* plot symmetric points */
          Plot_Point((int) xc+x,(int) yc+y,15);
          Plot_Point((int) xc-x,(int) yc+y,15);
          Plot_Point((int) xc+x,(int) yc-y,15);
          Plot_Point((int) xc-x,(int) yc-y,15);
          Plot_Point((int) xc+y,(int) yc+y,15);
          Plot_Point((int) xc-y,(int) yc+x,15);
          Plot_Point((int) xc+y,(int) yc-x,15);
          Plot_Point((int) xc-y,(int) yc-x,15);
          /* compute new x and y */
          xtemp = x;
          x = (xtemp*cosval) - (y*sinval);
          y = (y*cosval) + (xtemp*sinval);
    }
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}
```

```
int Get_Max_Y(void)
{
    return(getmaxy());
}
```

ในการวาดรูปวงกลมเราอาจจะใช้วิธีวาดส่วนของเส้นตรง (line segment) มาประกอบกันเป็นวงกลมก็ได้และยิ่งมีจำนวนส่วนของเส้นตรงมาก ก็จะได้วงกลมที่มีเส้นรอบวงโค้งมากขึ้น รูปที่ 2.10 แสดงวงกลม 2 รูป ซึ่งวาดโดยใช้ส่วนของเส้นตรงมาเชื่อมต่อกัน รูปที่ 2.10(ก) ใช้จำนวนเส้นตรงน้อย จึงดูไม่ค่อยเป็นวงกลม ส่วนรูปที่ 2.10(ข) ใช้จำนวนเส้นตรงมากขึ้นจึงดูเป็นวงกลมมากขึ้น



(ก) (ข)

รูปที่ 2.10 วงกลมซึ่งมีจำนวนเส้นตรงประกอบต่างกัน

โปรแกรมที่ 2.10 การสร้างรูปวงกลมโดยวิธีวาดส่วนของเส้นตรงมาประกอบกัน

```
/* A C program that demonstrates how to draw a circle using
 * line segments, assume that aspect ratio = 1
*/


#include <graphics.h>
#include <math.h>


#define TWO_PI 6.28318
```

```
void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Draw_Circle_2(int xc,int yc,int radius,int line_seg);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);

main()
{
    int i;

    Initialize_Graphics_Mode();

    for (i=1; i<=4; i++)
        Draw_Circle_2(Get_Max_X()/2,Get_Max_Y()/2,40*i,8*i);
    Out_Text_XY(Get_Max_X()/2-110,Get_Max_Y()-50,
        "A circle of various line segment");

    getchar();
    Close_Graphics_Mode();
    return(0);
}
void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
```

```
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
          printf("Graphics error : %s\n", grapherrormsg(errorcode));
          printf("Press any key to halt : ");
          getch();
          exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Circle_2(int xc,int yc,int radius,int line_seg)
{
    double dtheta,   /* angle increment */
           cosval,sinval, /* values of cos(dtheta) and sin(dtheta) */
           x,y,   /* computed point */
           xtemp;   /* old value of x */
    int    count;
    /* initialize */
    dtheta = TWO_PI/line_seg;
    cosval = cos(dtheta); sinval = sin(dtheta);
    /* initial coordinates */
    x = 0; y = radius;
    Move_To((int) xc+x,(int) yc+y);
    for(count=1; count<=line_seg; count++) {
           xtemp = x;
           x = (x*cosval) - (y*sinval);
```

```
            y = (y*cosval) + (xtemp*sinval);
            Draw_Line_To((int) xc+x,(int) yc+y );
    }
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}
```

2.5.3 คำสั่งสำหรับการสร้างวงกลม

ระบบกราฟิกส์โดยทั่วไปจะมีคำสั่งสำหรับสร้างรูปวงกลมเตรียมไว้ให้แล้ว ผู้ใช้เพียงแต่กำหนดจุดศูนย์กลางและรัศมีของวงกลม สำหรับคำสั่งที่ใช้ในการเขียนโปรแกรมตัวอย่างในหนังสือเล่มนี้ จะใช้เป็น

Draw_Circle(x,y,radius)

โดยที่ (x,y) เป็นจุดพิกัดที่ใช้เป็นจุดศูนย์กลางของวงกลม และ radius เป็นรัศมีของวงกลมซึ่งต่อไปจะใช้คำสั่งนี้โดยเรียกใช้คำสั่ง circle ในเทอร์โบซี

โปรแกรมที่ 2.11 การสร้างรูปวงกลมโดยใช้คำสั่ง Draw_Circle โดยเรียกใช้คำสั่ง circle ซึ่งมีอยู่ในเทอร์โบซีแล้ว

```
/* A C program that demonstrates how to draw a circle using
 * command circle in TURBO C, assume that aspect ratio = 1
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Draw_Circle(int xc,int yc,int radius);

main()
{
    int i;

    Initialize_Graphics_Mode();
```

```
    for (i=1; i<=15; i++)
        Draw_Circle(Get_Max_X()/2,Get_Max_Y()/2,i*10);
    Out_Text_XY(Get_Max_X()/2-150,Get_Max_Y()-50,
        "A demonstration of circles drawing");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
```

```
}

void Draw_Circle(int xc,int yc,int radius)
{
    circle(xc,yc,radius);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}
```

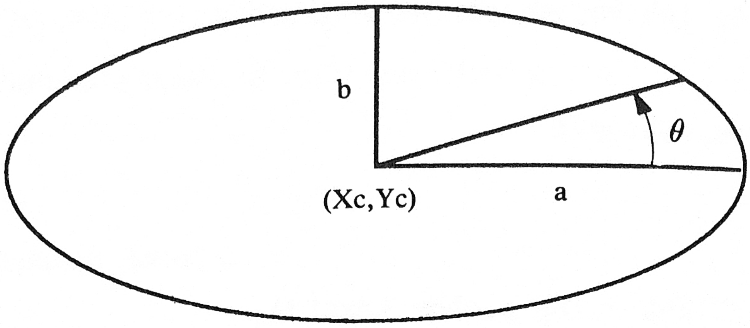
2.6 การวาดรูปวงรี

วงรี เป็นรูปทรงที่แปลงมาจากวงกลม โดยการยืดวงกลมออกไปในทิศทางหนึ่ง ในที่นี้จะพิจารณาเฉพาะวงรี ซึ่งยืดออกไปตามแนวแกน x หรือแกน y เท่านั้น เพื่อสะดวกในการเขียนสมการแทนวงรี

สมการวงรีเชิงมุมสำหรับวงรีในแกน x หรือแกน y และมีจุดศูนย์กลางอยู่ที่พิกัด (Xc,Yc) สามารถเขียนได้ดังนี้

$$\left.\begin{aligned} x &= Xc + a\cos(\theta) \\ y &= Yc + b\sin(\theta) \end{aligned}\right\} \qquad \text{.......... 2.10}$$

โดยที่ มุม θ จะมีค่าอยู่ระหว่าง 0 ถึง 2π ส่วนค่า a และ b กำหนดตามรูปที่ 2.11



รูปที่ 2.11 วงรีและการกำหนดพารามิเตอร์ของวงรี

ค่าของ a และ b จะมีผลต่อรูปร่างของวงรี กล่าวคือ ถ้า $b > a$ วงรีจะยืดออกตามแกน y แต่ถ้า $a > b$ วงรีก็จะยืดออกตามแกน x

จากสมการที่ 2.10 เราสามารถเขียนใหม่ให้แสดงความสัมพันธ์ระหว่างจุดพิกัด 2 จุด (x1,y1) กับ (x2,y2) ซึ่งเป็นจุดที่เรียงต่อกันได้ดังนี้

$$\left.\begin{aligned} x2 &= x1\ \cos(d\theta) - \left(\frac{a}{b}\right) y1\ \sin(d\theta) \\ y2 &= y1\ \cos(d\theta) - \left(\frac{b}{a}\right) x1\ \sin(d\theta) \end{aligned}\right\} \quad \ldots\ldots\ldots 2.11$$

สำหรับการสร้างรูปวงรีบนจอภาพคอมพิวเตอร์อาจทำได้ง่ายโดยการเปลี่ยนค่าอัตราส่วนแอสเป็กต์ หรือการใช้สมการที่ 2.11 ก็ได้ ซึ่งก็จะคล้าย ๆ กับการสร้างรูปวงกลม อย่างไรก็ตามในระบบกราฟิกส์ทั่วไปก็**จะมีคำสั่งหรือโปรแกรมย่อยสำหรับการวาดรูปวงรีไว้ให้ใช้ได้** สำหรับใน**หนังสือเล่มนี้จะใช้คำสั่ง**

Draw_Ellipse(x,y,stangle,endangle,xradius,yradius)

โดยที่ (x,y) เป็นจุดศูนย์กลางของวงรี stangle และ endangle **เป็นมุมเริ่มต้น**วาดส่วนโค้งและมุมสิ้นสุดที่จะวาด โดยที่มุม 0 จะอยู่ที่แกน x ด้านขวาของจุดศูนย์กลาง แล้วหมุนทวนเข็มนาฬิกาขึ้นไปก็จะเป็นมุมต่าง ๆ เช่น มุม 90 องศา คือมุมตามแนวแกน y นั่นเอง ถ้าเริ่มต้นที่ 0 องศา สิ้นสุดที่ 360 องศา ก็คือวาดรูปวงรีนั่นเอง แต่ถ้าวาดไม่ถึง 360 องศา ก็จะได้ส่วนโค้งแบบต่าง ๆ สำหรับ xradius และ yradius คือ รัศมีตามแกน x และ แกน y

โปรแกรมที่ 2.12 การสร้างรูปวงรีโดยใช้คำสั่งที่มีอยู่ในเทอร์โบซี

```
/* A C program that demonstrates how to draw an ellipse using
 * command ellipse in TURBO C, assume that aspect ratio = 1
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Draw_Ellipse(int xc,int yc,int stangle,int endangle,
                  int xradius,int yradius);

main()
{
    Initialize_Graphics_Mode();

    Draw_Ellipse(Get_Max_X()/2,Get_Max_Y()/2,0,360,200,100);
    Draw_Ellipse(Get_Max_X()/2,Get_Max_Y()/2,0,360,100,200);
    Draw_Ellipse(Get_Max_X()/2,Get_Max_Y()/2,0,90,40,80);
    Draw_Ellipse(Get_Max_X()/2,Get_Max_Y()/2,0,180,80,40);
    Out_Text_XY(Get_Max_X()/2-140,Get_Max_Y()-50,
          "A demonstration of ellipses drawing");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
```

```
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Ellipse(int xc,int yc,int stangle,int endangle,
                  int xradius,int yradius)
{
    ellipse(xc,yc,stangle,endangle,xradius,yradius);
}

void Out_Text_XY(int x,int y,char text[])
{
```

```
    outtextxy(x,y,text);
}
int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}
```

2.7 การประยุกต์ใช้งาน

2.7.1 ภาพรูปเกลียว

รูปเกลียว (spirals) เป็นรูปที่ดัดแปลงมาจากรูปวงกลม เช่น รูปเกลียวสามารถสร้างจากการวาดรูปวงกลมโดยให้ค่ารัศมีเปลี่ยนค่าไปเรื่อย ๆ

โปรแกรมที่ 2.13 การสร้างรูปเกลียวโดยใช้โปรแกรมการสร้างรูปวงกลม

```
/* A C program that demonstrates how to draw spirals */

#include <graphics.h>
#include <math.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Spirals(void);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
```

```
main()
{
    Initialize_Graphics_Mode();

    Spirals();
    Out_Text_XY(Get_Max_X()/2-110,Get_Max_Y()-100,
         "A demonstration of spirals");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}
```

```
void Close_Graphics_Mode(void)
{
    closegraph();
}

void Spirals(void)
{
    double theta,  /* angle */
           xi,yi,  /* initial point */
           x,y,    /* subsequent points */
           xc,yc,  /* center */
           radius; /* changing radius */

    /* initialize */
    xc = Get_Max_X()/2; yc = Get_Max_Y()/2;
    radius = 10;
    theta = 0;
    /* compute initial point */
    xi = radius*cos(theta); yi = radius*sin(theta);
    /* move to initial point */
    Move_To((int) xc+xi,(int) yc+yi);
    /* compute remaining points */
    while (radius < 100) {
          theta = theta + 0.1; /* increase theta */
          radius = radius + 0.1; /* increase radius */
          /* subsequent points */
          x = radius*cos(theta); y = radius*sin(theta);
          /* draw from previous point */
          Draw_Line_To((int) xc+x,(int) yc+y);
    }
}

void Draw_Line_To(int x,int y)
```

```
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}
int Get_Max_Y(void)
{
    return(getmaxy());
}
```

2.7.2 ภาพมอร์แพทเทิร์น

เป็นภาพที่เกิดจากการแสดงรูปในแบบหนึ่งซ้ำ ๆ กัน โดยให้ตำแหน่งที่เริ่มต้นแสดงรูปนั้นห่างกันเป็นระยะทางน้อย ๆ หรือมุมที่แสดงเป็นมุมที่ต่างกันน้อย ๆ ก็ได้ ผลที่ได้จะเป็นภาพซึ่งเรียกกันว่า ภาพมอร์แพทเทิร์น (Moire pattern)

โปรแกรมที่ 2.14 การสร้างภาพมอร์แพทเทิร์นโดยใช้โปรแกรมการสร้างรูปวงกลมเป็นรูปหลัก

```
/* A C program that demonstrates how to draw moire pattern */

#include <graphics.h>

#define INCREMENT 2
#define MAXRADIUS 100

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Moire_Pattern(void);
void Draw_Circle(int xc,int yc,int radius);

main()
{
    Initialize_Graphics_Mode(); /* a function in program 2.1 */

    Moire_Pattern();
    Out_Text_XY(Get_Max_X()/2-150,Get_Max_Y()-100,
         "A demonstration of Moire Pattern");

    getchar();
    Close_Graphics_Mode(); /* a function in program 2.1 */
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
```

```
    int gdriver = DETECT, gmode, errorcode;
    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
           printf("Graphics error : %s\n", grapherrormsg(errorcode));
           printf("Press any key to halt : ");
           getch();
           exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Moire_Pattern(void)
{
    int xc,yc,   /* center point */
        x,y,     /* points for lines */
        radius,i;

    xc = Get_Max_X()/2; yc = Get_Max_Y()/2;
    for(i=1; i<=5; i++) {
        radius = 10;
        while (radius < MAXRADIUS) {
            Draw_Circle(xc,yc,radius);
```

```
                radius += INCREMENT;
            }
            xc -= 10;
        }
}

void Draw_Circle(int xc,int yc,int radius)
{
    circle(xc,yc,radius);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}
```

2.8 สรุป

ภาพต่าง ๆ อาจจะสร้างขึ้นได้จากจุดและเส้น หรือภาพกราฟิกส์เบื้องต้นอื่น ๆ เช่น วงรี วงกลม เป็นต้น ความเร็วของการแสดงภาพบนจอจะขึ้นกับความเร็วของฮาร์ดแวร์และความเร็วของซอฟต์แวร์ที่ใช้ เช่น ซอฟต์แวร์ควรใช้การบวกจำนวนเต็มมากกว่าที่จะใช้การคูณหรือการหารเลขจำนวนจริงซึ่งช้ากว่า ดังนั้นในการเขียนโปรแกรมควรใช้การคำนวณเลขจำนวนเต็มเสมอถ้าเป็นไปได้

3 หน้าต่าง ช่องแสดงภาพและการขริบภาพ

3.1 บทนำ

ในบทที่ผ่านมา การสร้างภาพเส้นหรือรูปทรงต่าง ๆ จะถูกจำกัดให้อยู่ภายในขอบเขตของขนาดของจอภาพ เช่น จอภาพมีขนาด 640 × 350 พิกเซล หมายความว่าพิกัดสูงสุดที่ใช้ได้คือจุดที่ตำแหน่ง (639,349) ถ้าเกินจากนี้ก็จะใช้ไม่ได้

สำหรับภาพกราฟิกส์ที่ประยุกต์ใช้ในงานต่าง ๆ ขอบเขตเช่นนี้ไม่เพียงพอแก่ความต้องการของงาน ตัวอย่างเช่น ในงานเกี่ยวกับการออกแบบวงจรไฟฟ้า ซึ่งมีส่วนประกอบจำนวนมากไม่สามารถวาดวงจรทั้งหมดให้อยู่ภายในหน้าจอเดียวได้ ดังนั้นระบบกราฟิกส์จะต้องมีความสามารถในการเก็บภาพไม่จำกัดแค่เพียงขนาดจอภาพ และสามารถเลือกภาพส่วนใดส่วนหนึ่งที่ต้องการจากภาพทั้งหมดมาแสดงบนจอภาพได้ ความสามารถเช่นนี้ทำให้นักเขียนโปรแกรมและผู้ใช้งานโปรแกรมกราฟิกส์ควบคุมการแสดงภาพบนจอภาพได้ตามที่ต้องการ

ในบทนี้จะกล่าวถึงวิธีการต่าง ๆ ที่ใช้ในการควบคุมการแสดงผลบนจอภาพ พร้อมทั้งตัวอย่างโปรแกรม

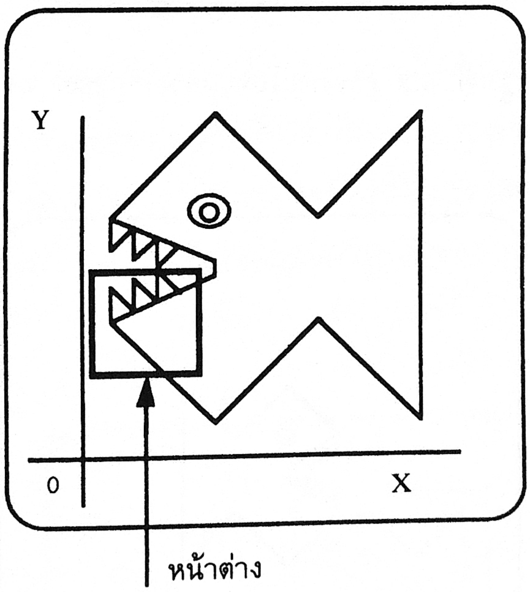
3.2 ภาพบนจอภาพ

วัตถุต่าง ๆ ในความเป็นจริงจะมีขนาดของมันเอง และมีการกำหนดหน่วยต่าง ๆ ขึ้นมาใช้แทนขนาดของวัตถุ เพื่อทำให้เราสามารถบอกความแตกต่างของขนาดของวัตถุได้ และยังสามารถใช้หน่วยเหล่านี้ระบุตำแหน่งของวัตถุในธรรมชาติได้ด้วย ตัวอย่างเช่น หน่วยเมตร สามารถใช้เป็นหน่วยบอกขนาดของวัตถุ และใช้เป็นหน่วยบอกขนาดระยะทางด้วย ในการนำภาพวัตถุมาแสดงบนจอภาพ เราควรจะสามารถเลือกระบบพิกัดซึ่งทำให้ผู้ใช้สามารถอธิบายตำแหน่งของวัตถุได้เช่น

เดียวกับการอธิบายตำแหน่งของวัตถุในธรรมชาติ หลังจากที่เลือกระบบพิกัดที่เหมาะสมแล้ว เราก็จะใช้วิธีการต่าง ๆ เพื่อแปลงภาพให้ไปแสดงอยู่ในบริเวณของจอภาพได้ นั่นคือแปลงให้อยู่ในระบบพิกัดของจอภาพนั่นเอง

3.2.1 หน้าต่างและการขริบภาพ

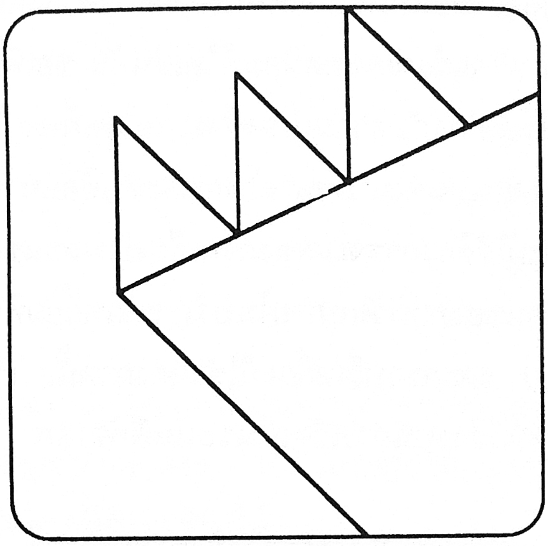
ระบบพิกัดโลก (world coordinate system) เป็นระบบที่เราเลือกใช้อธิบายตำแหน่งของภาพวัตถุในธรรมชาติ ระบบนี้จะไม่ขึ้นกับขนาดพิกัดของจอภาพ ดังนั้นช่วงตัวเลขที่ใช้จะเป็นเท่าไรก็ได้ จะมีค่าเป็นบวกหรือลบ หรือแม้แต่เลขทศนิยมก็ได้เช่นกัน ขอเพียงแต่ให้สามารถช่วยอธิบายตำแหน่งภาพวัตถุได้ถูกต้องใกล้เคียงกับความเป็นจริงมากที่สุดก็พอ ในบางครั้งภาพสมบูรณ์ของวัตถุที่ใช้ระบบพิกัดโลกอธิบายตำแหน่งนั้น มีขนาดใหญ่และซับซ้อนมาก จนไม่สามารถแสดงให้เห็นอย่างชัดเจนได้บนจอภาพ และผู้ใช้ต้องการจะแสดงภาพเพียงบางส่วนของวัตถุเท่านั้น ความสามารถในการแสดงภาพเพียงบางส่วนของภาพวัตถุภายในบริเวณสี่เหลี่ยมที่กำหนดให้ เราเรียกว่า *การทำช่องหน้าต่าง (windowing)* และกรอบสี่เหลี่ยมที่ใช้แสดงภาพนั้น เรียกว่า *หน้าต่าง (window)* รูปที่ 3.1 แสดงการทำช่องหน้าต่างบนภาพวัตถุในระบบพิกัดโลก



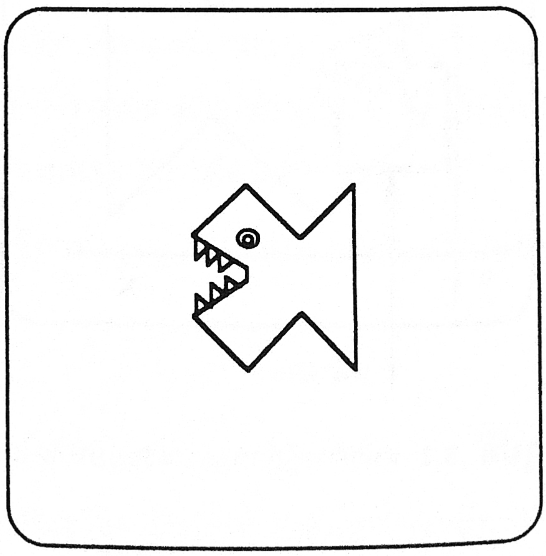
รูปที่ 3.1 หน้าต่างที่กำหนดบนระบบพิกัดโลก

ส่วนต่าง ๆ ของภาพจะถูกนำมาแสดงบนจอภาพได้โดยการนำหน้าต่างไปวางไว้ตรงส่วนนั้นของภาพในระบบพิกัดโลก การปรับเปลี่ยนขนาดของหน้าต่างจะทำให้เกิดการย่อ การขยาย หรือการบิดเบี้ยวภาพซึ่งอยู่ภายในหน้าต่างได้ รูปที่ 3.2 แสดงภาพภายในหน้าต่างจากรูปที่ 3.1 โดยให้ขนาดของหน้าต่างเท่ากับขนาดของจอภาพ ซึ่งมีผลทำให้เกิดการขยายภาพที่อยู่ภายในหน้าต่าง

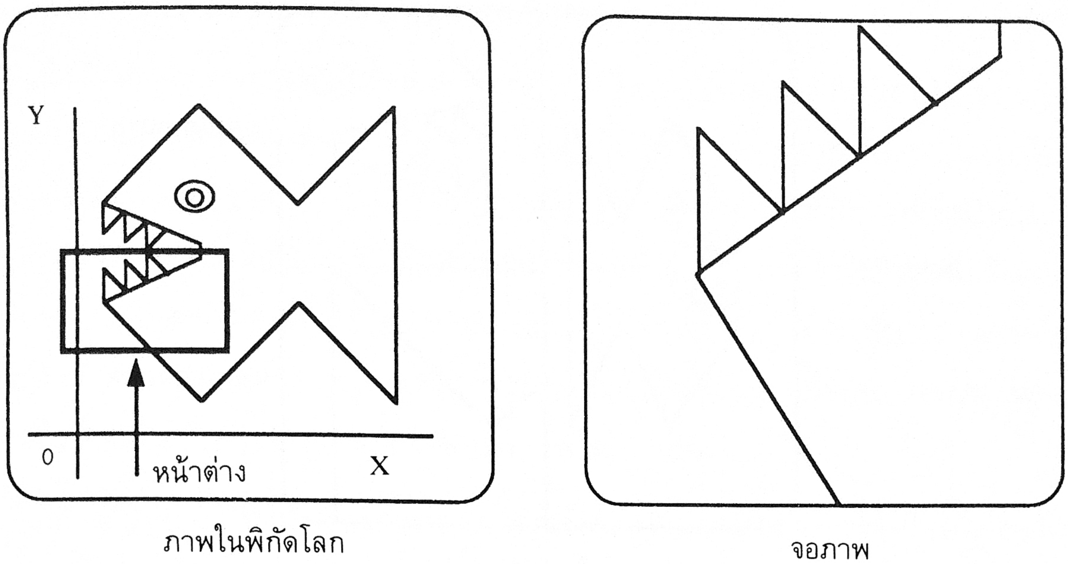
แต่ถ้าให้หน้าต่างขนาดใหญ่กว่าภาพมาก ๆ มาล้อมรอบภาพทั้งหมดแล้วนำมาแสดงบนจอภาพ จะมีผลทำให้เกิดการย่อขนาดของภาพลงมาดังแสดงในรูปที่ 3.3 การนำหน้าต่างขนาดต่างกันไปวางบนส่วนของภาพส่วนเดียวกัน เวลาแสดงภาพบนจอภาพจะได้ภาพที่ขนาดต่างกันด้วย ดังแสดงในรูปที่ 3.4 เปรียบเทียบกับรูปที่ 3.2 ส่วนของภาพที่อยู่นอกเหนือจากกรอบของหน้าต่าง จะไม่ถูกนำมาแสดงบนจอภาพ เราเรียกขั้นตอนการตัดส่วนเกินออกนี้ว่า ***การขริบภาพ (clipping)***



รูปที่ 3.2 ภาพภายในหน้าต่างจากรูปที่ 3.1



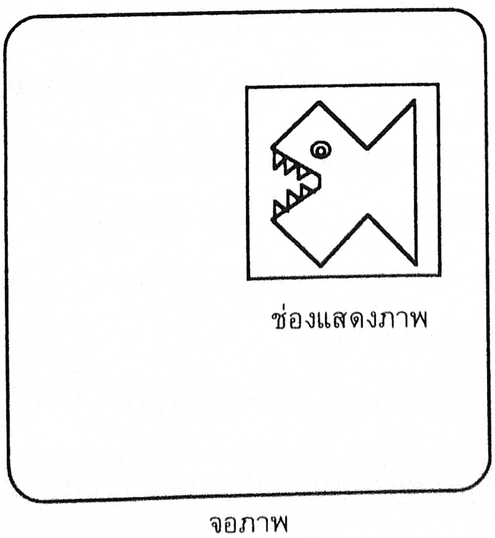
รูปที่ 3.3 ภาพที่ได้เมื่อทำหน้าต่างขนาดใหญ่มาก ๆ รอบวัตถุ



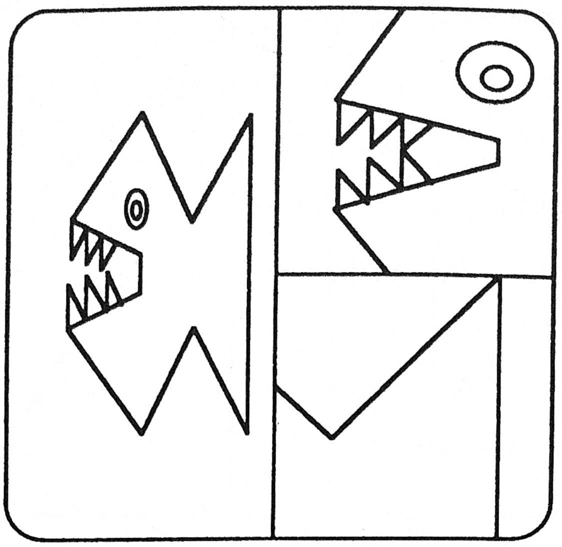
รูปที่ 3.4 ขนาดของหน้าต่างที่ใช้จะทำให้ภาพภายในหน้าต่างไม่เหมือนกัน

3.2.2 ช่องแสดงภาพ

บางครั้งเราต้องการแสดงส่วนของภาพหลาย ๆ ส่วนบนจอภาพเดียวกัน โดยให้เห็นทุกส่วนพร้อม ๆ กัน เราทำได้โดยแบ่งจอภาพเป็นส่วน ๆ แต่ละส่วนของจอภาพนั้น แสดงแต่ละส่วนของภาพที่ต้องการ เนื้อที่สี่เหลี่ยมบนจอภาพซึ่งมีส่วนของภาพในหน้าต่างที่กำหนดแสดงอยู่นั้น เราเรียกว่า *ช่องแสดงภาพ (viewport)* ดังแสดงในรูปที่ 3.5 การใช้หน้าต่างเดียวกันโดยมีช่องแสดงภาพหลาย ๆ ช่องทำให้เราสามารถแสดงภาพวัตถุเดียวกันในหลาย ๆ ส่วนของจอภาพได้ในเวลาเดียวกัน การเลือกหน้าต่างและช่องแสดงภาพแตกต่างกันก็ทำให้เกิดการแสดงภาพบนจอภาพได้หลายแบบ รูปที่ 3.6 แสดงหน้าต่าง 1 หน้าต่างซึ่งมีช่องแสดงภาพ 3 ช่อง ภาพในช่องแสดงภาพอาจจะเป็นภาพเดียวกันหรือไม่ก็ได้



รูปที่ 3.5 ช่องแสดงภาพบนส่วนหนึ่งของจอภาพ



รูปที่ 3.6 ช่องแสดงภาพ 3 ช่องบนจอภาพหรือหน้าต่างเดียว

3.2.3 การกวาดหาภาพ

การเลื่อนหน้าต่างไปในส่วนต่าง ๆ ของภาพซึ่งอยู่ในระบบพิกัดโลก ทำให้ผู้ใช้งานสามารถเลือกดูส่วนต่าง ๆ ของภาพได้ ผู้ใช้สามารถควบคุมและกำหนดตำแหน่งและขนาดของหน้าต่างได้โดยใช้อุปกรณ์นำเข้าที่เหมาะสม การเลื่อนหน้าต่างนี้เรียกว่า *การกวาดหาภาพ (panning)* ซึ่งการจัดการทำได้ทั้งทางฮาร์ดแวร์และซอฟต์แวร์

สำหรับการจัดการทางด้านฮาร์ดแวร์ ภาพวัตถุทั้งหมดจะถูกเก็บไว้ในหน่วยความจำที่เรียกว่าเฟรมบัฟเฟอร์ ซึ่งจะมีขนาดใหญ่กว่าขนาดของจอภาพและกำหนดให้หน้าต่างมีขนาดเท่ากับขนาดของจอภาพ แล้วให้หน้าต่างนี้เลื่อนไปตามหน่วยความจำ ขณะที่เลื่อนไปก็ให้แสดงข้อมูลที่เก็บไว้ซึ่งก็คือภาพออกมาทางจอภาพ นั่นคือเราสามารถเลือกแสดงส่วนใดส่วนหนึ่งของภาพวัตถุได้โดยให้หน้าต่างไปอยู่ตรงตำแหน่งซึ่งเก็บภาพส่วนนั้นไว้ แล้วส่งภาพออกมาทางจอภาพ ถ้าหน่วยความจำมีราคาถูก วิธีนี้ก็เหมาะที่จะนำมาใช้งานมากเนื่องจากสามารถแสดงผลได้อย่างรวดเร็ว

สำหรับการกวาดหาภาพโดยใช้ซอฟต์แวร์นั้นจะมีลำดับดังนี้

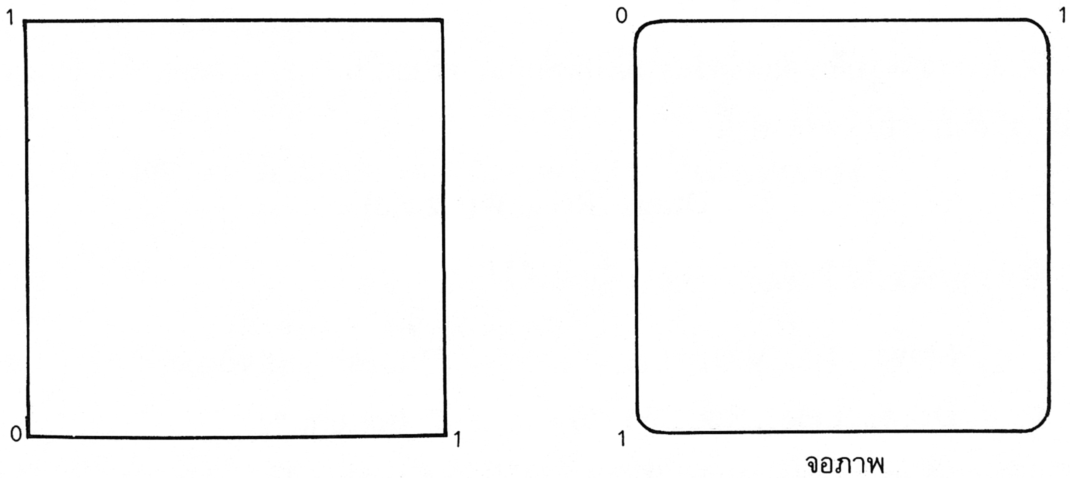
- แสดงภาพในหน้าต่าง
- ลบภาพในหน้าต่าง
- เลื่อนหน้าต่างไปยังตำแหน่งใหม่
- แสดงภาพในหน้าต่างซึ่งมีส่วนของภาพอันใหม่

ขั้นตอนข้างต้นจะต้องมีความเร็วในการทำงานมากเพียงพอที่จะทำให้การกวาดหาภาพเป็นไปอย่างราบรื่น

3.2.4 ระบบพิกัดที่ไม่ขึ้นกับขนาดของจอภาพ

เนื่องจากจอภาพมีหลายขนาดต่าง ๆ กันไป การระบุขนาดของช่องแสดงภาพโดยใช้ขนาดของจอภาพเป็นตัวกำหนดทำให้การแสดงภาพขึ้นอยู่กับขนาดของจอภาพนั้น ๆ เมื่อไปใช้จอภาพขนาดอื่น ๆ ก็จะต้องแก้ไขให้เข้ากับจอภาพขนาดนั้น ๆ อีก ดังนั้นเพื่อแก้ปัญหานี้จึงมีระบบพิกัดซึ่งไม่ขึ้นกับขนาดของจอภาพขึ้นเรียกว่า ***พิกัดนอร์มอลไลซ์ดีไวซ์** (normalized device coordinates หรือ NDC)* ระบบพิกัดนี้จะให้ค่าแกน x และแกน y มีค่าอยู่ระหว่าง 0 กับ 1 มีจุดศูนย์กลางอยู่ตรงมุมล่างซ้ายดังแสดงในรูปที่ 3.7 แต่ในทางปฏิบัติแล้วจุด (0,0) ของจอภาพจะอยู่ที่มุมบนซ้าย ดังนั้นเราจึงต้องมีโปรแกรมแปลงพิกัดให้แสดงภาพได้ถูกต้อง

ระบบนี้ทำให้ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมกราฟิกส์ได้โดยไม่ต้องคำนึงถึงความละเอียดของจอภาพ สำหรับการแสดงผลภาพ ระบบการแสดงผล จะทำการแปลง NDC ให้ไปเป็นพิกัดของจอภาพขณะนั้น โดยอาจจะใช้วิธีทางฮาร์ดแวร์หรือโปรแกรมทางซอฟต์แวร์ก็ได้



รูปที่ 3.7 ระบบพิกัดนอร์มอลไลซ์ดีไวซ์

3.3 การวาดภาพในพิกัดโลก

ภาพ 2 มิติในระบบพิกัดโลกประกอบไปด้วยจุดและเส้น มีเพียงบางส่วนของภาพซึ่งอยู่ภายในหน้าต่างเท่านั้นที่ปรากฏบนช่องแสดงภาพ ในตอนนี้จะกล่าวถึงคำสั่งต่าง ๆ ที่ใช้ในการวาดจุด เส้นและภาพอื่น ๆ บางคำสั่ง ซึ่งใช้วาดภาพในระบบพิกัดโลกได้

3.3.1 คำสั่งสำหรับสร้างจุดและเส้นในระบบพิกัดโลก

แม้ว่าในการวาดภาพลงบนจอภาพเราจะใช้ระบบพิกัดของจอภาพอยู่ก็ตาม แต่เราอาจจะคิดว่าเรากำลังเคลื่อนที่ไปในระบบพิกัดโลกก็ได้ โดยการลากเส้นเชื่อมจุดต่าง ๆ ในพิกัดและคำสั่งต่าง ๆ ต่อไปนี้ เรากำหนดให้เป็นคำสั่งที่เกี่ยวข้องกับการสร้างจุดและเส้นในระบบพิกัดโลกดังนี้

Plot__Point__W(x,y,c)	สำหรับสร้างจุดสี c ที่ตำแหน่ง (x,y)
Move__To__W(x,y)	สำหรับเลื่อนจุด CP ไปยังจุด (x,y)
Move__Rel__W(dx,dy)	สำหรับเลื่อนจุด CP จากตำแหน่งปัจจุบันนั่นคือ จุด (x,y) ไปยังตำแหน่งใหม่คือ จุด (x+dx,y+dy)
Draw__Line__To__W(x,y)	สำหรับลากเส้นจากจุด CP ไปยังจุด (x,y)
Draw__Line__Rel__W(dx,dy)	สำหรับลากเส้นจากจุด (x,y) ซึ่งเป็นจุด CP ไปยังตำแหน่งใหม่คือ จุด (x+dx,y+dy)
Draw__Rec__W(x1,y1,x2,y2)	สำหรับสร้างรูปสี่เหลี่ยมโดยกำหนดจุดพิกัดที่มุมบนซ้ายหรือมุมล่างซ้ายคือ (x1,y1) และมุมล่างขวาหรือมุมบนขวา (x2,y2)

ตัวอย่าง

ต้องการวาดรูปสี่เหลี่ยมจัตุรัสซึ่งมีพิกัดที่มุมล่างซ้ายเป็น (1,2) และแต่ละด้านมีขนาดเท่ากับ 1 หน่วย เราสามารถใช้คำสั่งดังนี้

```
Draw__Rec__W(1,2,2,3)
```

ซึ่งเราอาจจะใช้ลำดับคำสั่งต่อไปนี้แทนก็ได้

```
Move__To__W(1,2);            /* lower left corner */
Draw__Line__Rel__W(1,0)      /* bottom */
Draw__Line__Rel__W(0,1)      /* right side */
Draw__Line__Rel__W(-1,0)     /* top */
Draw__Line__Rel__W(0,-1)     /* left side */
```

จะเห็นว่าคำสั่งที่ใช้ก็เหมือนกับคำสั่งสำหรับสร้างจุดและเส้นที่ใช้ในระบบพิกัดจอภาพนั่นเอง เพียงแต่ในระบบพิกัดจอภาพ จุดพิกัดจะเป็นค่าเลขจำนวนเต็มบวกหรือศูนย์ เพื่อที่สามารถจะแสดงภาพบนจอภาพได้ทั้งหมด ส่วนพิกัดที่ใช้ในระบบพิกัดโลกจะเป็นเลขจำนวนเต็มใด ๆ ก็ได้

3.3.2 คำสั่งเกี่ยวกับการแสดงภาพบนจอภาพ

กรอบสี่เหลี่ยมที่นำไปวางบนภาพวัตถุในระบบพิกัดโลกเราเรียกว่า ช่องหน้าต่าง ส่วนกรอบสี่เหลี่ยมที่อยู่ในพื้นที่ของจอภาพซึ่งสามารถแสดงภาพเดียวกันกับภาพในช่องหน้าต่างเราเรียกว่า ช่องแสดงภาพ ในระบบกราฟิกส์ทั่วไปจะมีคำสั่งเพื่อกำหนดทั้งช่องหน้าต่างและช่องแสดงภาพ ซึ่งในที่นี้จะกำหนดคำสั่งดังนี้

```
Set_Window(left,bottom,right,top)
```

เป็นคำสั่งสำหรับกำหนดช่องหน้าต่างโดยที่ (left,bottom) เป็นค่าพิกัดมุมล่างซ้าย ส่วนค่าพิกัดมุมบนขวาคือ (right,top) และขนาดของค่าทั้ง 4 ค่านี้จะเป็นค่าในระบบพิกัดโลก

สำหรับการกำหนดช่องแสดงภาพ จะใช้คำสั่ง

```
Set_Viewport(left,bottom,right,top)
```

โดยที่ (left,bottom) เป็นค่าพิกัดมุมล่างซ้ายและ (right,top) เป็นค่าพิกัดมุมบนขวา และขนาดของค่าทั้ง 4 ค่านี้จะเป็นค่าในระบบพิกัดของจอภาพที่กำลังใช้งานอยู่ หรืออาจจะเป็นค่าซึ่งอยู่ระหว่าง 0 กับ 1 กรณีใช้พิกัดพิกัดนอร์มอลไลซ์ดีไวซ์

คำสั่ง 2 คำสั่งข้างต้นจะใช้คู่กัน ตัวอย่างเช่น เราต้องการแสดงส่วนของภาพส่วนหนึ่งซึ่งอยู่ในระบบพิกัดโลก เราก็นำช่องหน้าต่างไปวางตรงส่วนนั้นโดยใช้คำสั่ง

```
Set_Window(-60.5,41.25,-20.75,82.5)
```

จากนั้น เราจะนำภาพในช่องหน้าต่างมาแสดงบนจอภาพได้โดยใช้คำสั่งกำหนดช่องแสดงภาพ โดยใช้พิกัดนอร์มอลไลซ์ดีไวซ์ เช่น

```
Set_Viewport(0.5,0.8,0.7,1.0)
```

ซึ่งภาพที่ได้จะแสดงอยู่ในส่วนเล็ก ๆ ตรงมุมบนขวาของจอภาพ แต่ถ้าต้องการให้แสดงภาพเต็มจอภาพ เราจะต้องกำหนดช่องแสดงภาพดังนี้

```
Set_Viewport(0,0,1,1)
```

ในการเขียนโปรแกรม เราควรเรียกใช้คำสั่งทั้งสองนี้ก่อนการสร้างภาพกราฟิกส์เสมอ และการเปลี่ยนขนาดของช่องหน้าต่างหรือช่องแสดงภาพหลังการวาดภาพลงบนจอภาพแล้วจะไม่ทำให้ภาพที่วาดไว้แล้วเปลี่ยนแปลงไป สำหรับระบบกราฟิกส์บางระบบจะจัดการเรียกใช้คำสั่งทั้งสองนี้โดยอัตโนมัติเลย เราจึงไม่ต้องเรียกใช้อีก

การที่จะวาดภาพใหม่ได้นั้นเราอาจต้องการลบภาพเดิมออกก่อน ซึ่งทำได้โดยใช้คำสั่ง

Clear__Viewport

ซึ่งจะทำให้มีการล้างช่องแสดงภาพนั้น โดยทำให้ช่องแสดงภาพมีสีเดียวกับสีพื้นนั่นเอง

สำหรับในเทอร์โบซี คำสั่งกราฟิกส์เกี่ยวกับการแสดงภาพบนจอภาพจะมีเพียงการเตรียมช่องแสดงภาพ setviewport และการล้างช่องแสดงภาพ clearviewport เท่านั้น จะไม่มีคำสั่งในการเตรียมหน้าต่าง การกำหนดขนาดและตำแหน่งของช่องแสดงภาพก็จะใช้จุดพิกัดของจอภาพนั้นเองเป็นตัวกำหนด

โปรแกรมที่ 3.1 เป็นโปรแกรมแสดงการใช้คำสั่ง setviewport และ clearviewport ซึ่งในที่นี้เรากำหนดให้เป็นคำสั่ง Set__Viewport และ Clear__Viewport

```
/* A C program that demonstrates how to use command
 * Set_Viewport and Clear_Viewport
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line(int x1,int y1,int x2,int y2);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Set_Viewport(int lt,int tp,int rt,int bt);
void Clear_Viewport(void);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
```

```
main()
{
    Initialize_Graphics_Mode();

    /* viewport 1 */
    Draw_Rectangle(20,20,200,200);
    Set_Viewport(20,20,200,200);
    Out_Text_XY(5,5,"Viewport 1");
    Draw_Line(0,50,250,50);
    getchar();

    Set_Viewport(0,0,Get_Max_X(),Get_Max_Y());
    /*viewport 2 */
    Draw_Rectangle(300,20,500,200);
    Set_Viewport(300,20,500,200);
    Out_Text_XY(5,5,"Viewport 2");
    Draw_Line(0,50,250,50);
    getchar();
    Out_Text_XY(5,100,"Now clear this viewport");
    getchar();
    Clear_Viewport();
    getchar();

    Set_Viewport(0,0,Get_Max_X(),Get_Max_Y());
    /* viewport 3 */
    Draw_Rectangle(20,210,500,340);
    Set_Viewport(20,210,500,340);
    Out_Text_XY(5,5,"Viewport 3");
    Draw_Line(0,50,250,50);

    getchar();
    Close_Graphics_Mode();
    return(0);
```

```
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
            printf("Graphics error : %s\n", grapherrormsg(errorcode));
            printf("Press any key to halt : ");
            getch();
            exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}
```

```
void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}

void Set_Viewport(int lt,int tp,int rt,int bt)
{
    setviewport(lt,tp,rt,bt,1);
}

void Clear_Viewport(void)
{
    clearviewport();
}
```

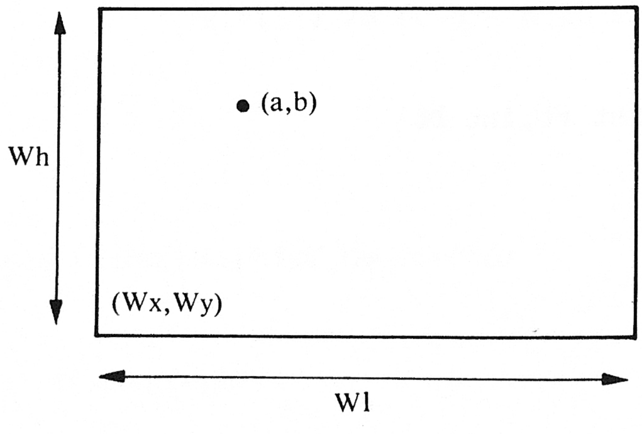
คำสั่ง Draw__Rectangle ใช้สำหรับลากเส้นแสดงกรอบของช่องแสดงภาพที่กำหนดขึ้น และจากโปรแกรมข้างต้น ก่อนที่จะกำหนดช่องแสดงภาพช่องที่ 2 เราต้องกำหนดช่องแสดงภาพให้

เท่ากับจอภาพเสียก่อน นั่นคือ Set__Viewport(0,0,Get__Max__X(),Get__Max__Y()) ถ้าไม่ทำเช่นนี้ ช่องแสดงภาพช่องที่ 2 จะอยู่ภายในช่องแสดงภาพช่องที่ 1 เนื่องจากช่องแสดงภาพช่องที่ 1 เป็นช่องแสดงภาพที่กำหนดไว้เป็นครั้งสุดท้าย และช่องแสดงภาพช่องที่ 2 อาจจะมีขนาดใหญ่กว่าช่องแสดงภาพช่องที่ 1 ก็ได้ ทำให้ไม่เห็นช่องแสดงภาพช่องที่ 2

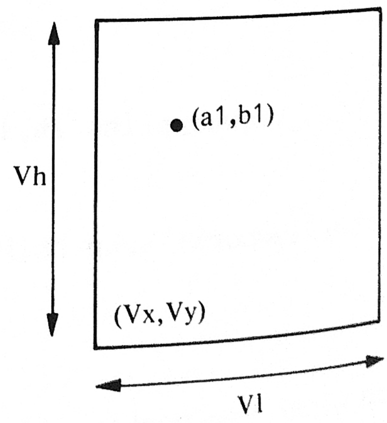
3.4 การแปลงภาพจากช่องหน้าต่างลงในช่องแสดงภาพ

ภาพซึ่งถูกกำหนดพิกัดในระบบพิกัดโลก เมื่อนำมาแสดงบนจอภาพผ่านทางช่องแสดงภาพ จะต้องมีการแปลงพิกัดจากพิกัดโลกไปเป็นพิกัดนอร์มอลไลซ์ดีไวซ์ นั่นคือการแปลงพิกัด (a,b) ในระบบพิกัดโลก ให้เป็นพิกัด (a1,b1) ในระบบพิกัดนอร์มอลไลซ์ดีไวซ์ จากนั้นก็แสดงภาพ การจะเปลี่ยนภาพบนจอภาพให้เป็นภาพอื่น ทำได้โดยการเปลี่ยนตำแหน่งของช่องหน้าต่างและ/หรือช่องแสดงภาพให้อยู่ในตำแหน่งใหม่ ทำการแปลงพิกัด แล้วแสดงภาพ

เราสามารถกำหนดช่องหน้าต่างและช่องแสดงภาพโดยใช้จุดพิกัดที่มุมล่างซ้ายกับความยาวทางแกน x และความสูงทางแกน y ได้ ตัวอย่างเช่น สำหรับช่องหน้าต่างมีจุดพิกัดที่มุมล่างซ้ายเป็น (Wx,Wy) และมีความสูงเป็น Wh ความกว้าง Wl (รูปที่ 3.8 (ก)) ส่วนช่องแสดงภาพนั้นมีพิกัดมุมล่างซ้ายเป็น (Vx,Vy) ความสูง Vh ความกว้าง Vl และความสูงมีขนาดเท่ากับความกว้างเพราะเป็น NDC (รูปที่ 3.8 (ข))



(ก) ช่องหน้าต่าง



(ข) ช่องแสดงภาพ

รูปที่ 3.8 การแปลงพิกัดช่องหน้าต่างไปเป็นพิกัดช่องแสดงภาพ

การแปลงพิกัดนี้จะเปลี่ยนพิกัดของจุด (a,b) ซึ่งอยู่ในช่องหน้าต่าง ให้กลายเป็นพิกัด NDC (a1,b1) ในช่องแสดงภาพ โดยมีสมการการแปลงดังนี้

1. ระยะทางจากขอบซ้ายของช่องแสดงภาพถึงจุด a1 หารด้วย ความกว้างของช่องแสดงภาพจะต้องเท่ากับ ระยะทางจากขอบซ้ายของช่องหน้าต่างถึงจุด a หารด้วย ความกว้างของช่องหน้าต่าง เขียนเป็นสมการดังนี้

$$\frac{(a1-Vx)}{Vl} = \frac{(a-Wx)}{Wl}$$

ซึ่งเขียนเป็นสมการการแปลงค่า a ให้เป็น a1 ได้ว่า

$$a1 = \frac{(Vl)(a-Wx)}{Wl} + Vx$$

2. ระยะทางจากขอบล่างของช่องแสดงภาพถึงจุด b1 หารด้วย ความสูงของช่องแสดงภาพจะต้องเท่ากับ ระยะทางจากขอบล่างของช่องหน้าต่างถึงจุด b หารด้วย ความสูงของช่องหน้าต่างเขียนเป็นสมการดังนี้

$$\frac{(b1-Vy)}{Vh} = \frac{(b-Wy)}{Wh}$$

ซึ่งเขียนเป็นสมการการแปลงค่า b ให้เป็น b1 ได้ว่า

$$b1 = \frac{(Vh)}{Wh}(b-Wy) + Vy$$

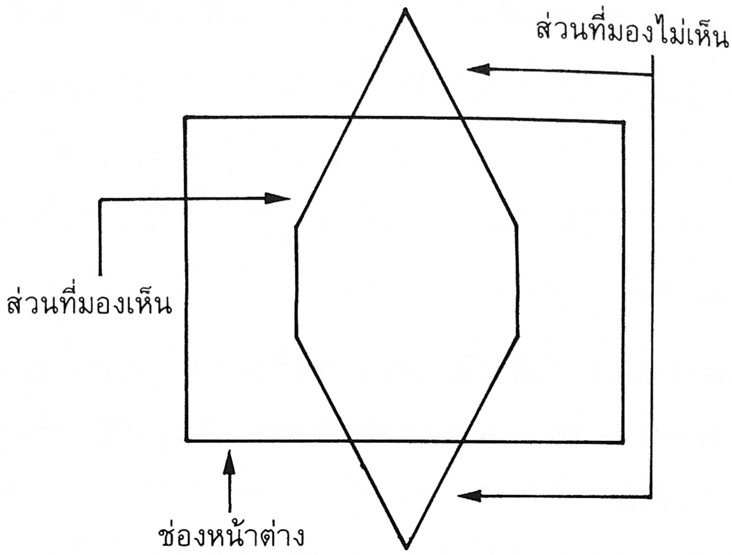
เนื่องจากเราต้องใช้สมการการแปลงพิกัดนี้ทุกครั้งที่มีการแปลงเกิดขึ้น เราจึงต้องหาวิธีลดเวลาในการคำนวณลงบ้าง เช่น จะเห็นว่าอัตราส่วน $\frac{Vl}{Wl}$ และ $\frac{Vh}{Wh}$ จะมีค่าคงที่สำหรับการแปลงทุกจุด ดังนั้นแทนที่จะทำการคำนวณค่าทั้งสองนี้ทุกครั้งในการแปลงพิกัด เราก็คำนวณหาค่าของมันเพียงครั้งเดียวในตอนเริ่มต้นก็พอ

3.5 การขริบภาพ

เมื่อนำหน้าต่างไปวางลงบนภาพวัตถุ ส่วนที่เราจะเห็นได้ก็คือส่วนที่อยู่ภายในกรอบของช่องหน้าต่าง (หรือช่องแสดงภาพ) เท่านั้น (รูปที่ 3.9) จุดและเส้นในส่วนที่อยู่นอกกรอบ จะถูกตัดออกไปโดยใช้ ***การขริบภาพ*** แล้วค่อยส่งจุดและเส้นที่อยู่ภายในกรอบไปทำการแปลงเพื่อแสดงทางจอภาพ

3.5.1 การขริบจุด

วิธีการตรวจสอบว่าจุดอยู่ในกรอบของช่องหน้าต่างหรือช่องแสดงภาพหรือไม่ ทำได้โดยตรวจสอบว่า จุดนั้นมีค่าพิกัดอยู่ระหว่างขอบซ้ายและขอบขวาของกรอบ และอยู่ระหว่างขอบบนและขอบล่างของกรอบหรือไม่ เขียนอธิบายได้ดังนี้



รูปที่ 3.9 ภาพที่อยู่นอกกรอบของหน้าต่างจะต้องถูกทำให้มองไม่เห็นโดยใช้การขริบภาพ

X-ขอบซ้าย <= x <= X-ขอบขวา และ Y-ขอบล่าง <= y <= Y-ขอบบน

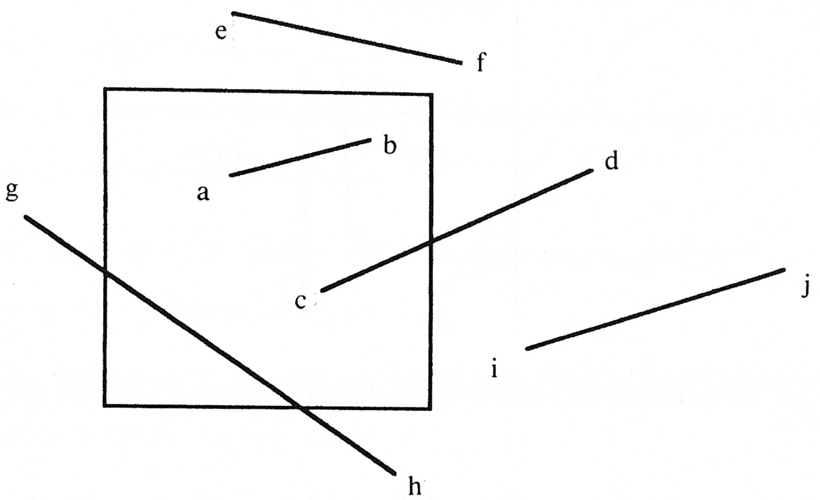
วิธีการขริบจุดนี้สามารถใช้ได้ถูกต้องแน่นอนเสมอ แต่ในทางปฏิบัติจะไม่ใช้กัน ทั้งนี้เนื่องมาจากเหตุผลสองประการคือ ประการแรก ภาพที่ต้องผ่านการขริบภาพอาจจะมีขนาดใหญ่และมีความซับซ้อนมาก ภาพก็จะประกอบไปด้วยจุดจำนวนมากทำให้เวลาในการตรวจสอบจุดมากตามไปด้วย จึงไม่เหมาะที่จะนำมาใช้ในระบบกราฟิกส์ ประการที่สอง ภาพอาจจะไม่ได้เก็บในลักษณะจุดทั้งหมดก็ได้ นั่นคืออาจจะแทนได้ด้วยจุดเพียงสองจุด จุดเริ่มต้นกับจุดปลาย เช่น เส้นตรงเส้นหนึ่งอาจจะเก็บในรูปของจุดจำนวนมากที่มาต่อกันเป็นเส้นตรง ซึ่งทำให้เปลืองเนื้อที่ในการเก็บมากหรือเก็บเฉพาะจุดเริ่มต้นกับจุดปลายแล้วมีตัวบอกว่าเป็นเส้นตรง ก็จะทำให้ใช้เนื้อที่ในการเก็บข้อมูลน้อยลง ถ้าเราเก็บภาพในลักษณะนี้การขริบภาพโดยตรวจสอบจุดก็จะทำได้ยาก เนื่องจากระหว่างจุดสองจุดอาจจะมีจุดเป็นจำนวนอนันต์ที่อยู่ระหว่างมัน ดังนั้นเราจึงต้องหาวิธีการอื่นมาใช้ตรวจสอบสำหรับการขริบภาพ ซึ่งได้แก่ การขริบเส้น

3.5.2 การขริบเส้น

รูปที่ 3.10 แสดงตำแหน่งต่าง ๆ ของเส้นซึ่งอาจจะอยู่ได้ทั้งภายนอกและภายในกรอบของหน้าต่างหรืออาจจะตัดกับเส้นกรอบหรืออยู่บนกรอบก็ได้ ตัวอย่างเช่น เส้น ab จะอยู่ภายในกรอบ

ส่วนเส้น ef และ ij อยู่ภายนอกกรอบ เส้น cd จุด c อยู่ในกรอบแต่จุด d อยู่นอกกรอบ ส่วนเส้น gh จุด g และ h อยู่นอกกรอบแต่ส่วนของเส้นบางส่วนอยู่ภายในกรอบ

เนื่องจากภาพที่ซับซ้อนอาจจะประกอบไปด้วยเส้นจำนวนมากจึงต้องมีวิธีการตรวจสอบสำหรับการขริบเส้นที่มีความเร็วสูง จึงสามารถแสดงภาพในหน้าต่างได้อย่างรวดเร็ว วิธีการที่ใช้กันมีอยู่หลายวิธี เช่น วิธีโคเฮน-ซูเธอร์แลนด์ (Cohen-Sutherland Algorithm) ซึ่งเป็นวิธีที่สามารถตรวจสอบว่าเส้นอยู่ภายนอกหรือภายในกรอบได้อย่างรวดเร็ว จึงเป็นวิธีที่นำไปใช้ในระบบกราฟิกส์มากที่สุดวิธีหนึ่ง



รูปที่ 3.10 ส่วนของเส้นในตำแหน่งต่าง ๆ เมื่อตัดกับหน้าต่าง

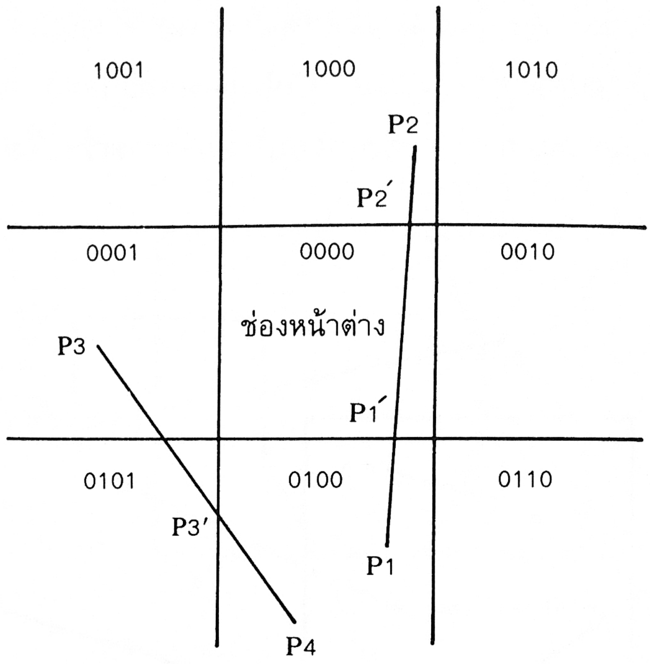
3.5.3 วิธีโคเฮน-ซูเธอร์แลนด์

สำหรับวิธีการนี้เราจะแบ่งระบบพิกัดโลกออกเป็นพื้นที่ 9 ส่วน โดยมีพื้นที่ตรงกลางเป็นช่องหน้าต่าง และพื้นที่อีก 8 ส่วนคือพื้นที่รอบ ๆ ช่องหน้าต่าง ดังแสดงในรูปที่ 3.11 แต่ละส่วนจะมีรหัสแทนเป็นเลขฐานสองจำนวน 4 บิต ดังที่กำหนดในรูปที่ 3.11 จุดพิกัดซึ่งตกอยู่ในพื้นที่ใดก็จะถูกกำหนดให้มีรหัสตามพื้นที่นั้น รหัสทั้ง 9 รหัสมีความหมายดังนี้

รหัส 1001 หมายถึง จุดอยู่ในตำแหน่งทาง บน-ซ้าย ของช่องหน้าต่าง
รหัส 1000 หมายถึง จุดอยู่ในตำแหน่งทาง บน ของช่องหน้าต่าง
รหัส 1010 หมายถึง จุดอยู่ในตำแหน่งทาง บน-ขวา ของช่องหน้าต่าง
รหัส 0001 หมายถึง จุดอยู่ในตำแหน่งทาง ซ้าย ของช่องหน้าต่าง
รหัส 0000 หมายถึง จุดอยู่ในตำแหน่ง ภายใน ของช่องหน้าต่าง
รหัส 0010 หมายถึง จุดอยู่ในตำแหน่งทาง ขวา ของช่องหน้าต่าง
รหัส 0101 หมายถึง จุดอยู่ในตำแหน่งทาง ล่าง-ซ้าย ของช่องหน้าต่าง

รหัส 0100 หมายถึง จุดอยู่ในตำแหน่งทาง ล่าง ของช่องหน้าต่าง

รหัส 0110 หมายถึง จุดอยู่ในตำแหน่งทาง ล่าง-ขวา ของช่องหน้าต่าง



รูปที่ 3.11 การแบ่งพื้นที่ของระบบพิกัดโลกเพื่อใช้การ**ขริบ**โดยวิธีโคเฮน-ซูเธอร์แลนด์

การอ้างถึงบิตแต่ละบิตในรหัสจะเป็นดังนี้ หลักหน่วยของรหัสจะเป็นบิต 1 หลักต่อมาก็เป็นบิต 2, บิต 3 และบิต 4 เช่น รหัส 1000 บิต 1,2 และ 3 เป็น 0 ส่วนบิต 4 เป็น 1 เป็นต้น แต่ละบิตสัมพันธ์กับตำแหน่งซึ่งเปรียบเทียบกับช่องหน้าต่างดังนี้

บิต 1 - ซ้าย (ถ้าค่าบิต 1 เป็น 1 หมายถึง อยู่ทางซ้ายของช่องหน้าต่าง)

บิต 2 - ขวา (ถ้าค่าบิต 2 เป็น 1 หมายถึง อยู่ทางขวาของช่องหน้าต่าง)

บิต 3 - ล่าง (ถ้าค่าบิต 3 เป็น 1 หมายถึง อยู่ทางล่างของช่องหน้าต่าง)

บิต 4 - บน (ถ้าค่าบิต 4 เป็น 1 หมายถึง อยู่ทางบนของช่องหน้าต่าง)

ค่าของแต่ละบิตในรหัสอาจจะเป็น 0 หรือ 1 ก็ได้ ขึ้นอยู่กับผลของการเปรียบเทียบค่าพิกัดปลายของเส้น (endpoint coordinate values) กับขอบเขตของช่องหน้าต่าง เช่น บิต 1 (ซ้าย) จะถูกกำหนดค่าให้เป็น 1 เมื่อค่า x ของค่าพิกัดปลายน้อยกว่าค่า x ของช่องหน้าต่าง ($x < xw_min$) อีก 3 บิตก็ใช้วิธีเปรียบเทียบทำนองเดียวกัน สำหรับภาษาซีซึ่งสามารถจัดการกับบิตได้ การกำหนดรหัสให้กับจุดพิกัดปลายของเส้นทำได้ดังนี้

1. คำนวณหาค่าผลต่างของค่าพิกัดปลายของเส้นกับขอบเขตของช่องหน้าต่าง
2. ใช้ผลลัพธ์ที่ได้ซึ่งอาจจะเป็นค่าบวกหรือค่าลบไปกำหนดค่า 1 หรือ 0 ให้กับบิตต่าง ๆ ของรหัส กล่าวคือ

- ถ้า x-xw__min < 0 แสดงว่าจุดพิกัดอยู่ทางซ้ายของช่องหน้าต่าง
  บิต 1 จะถูกกำหนดให้เป็นค่า 1
  ถ้า x-xw__min >= 0 แสดงว่าจุดพิกัดอยู่ภายในหรือบนขอบเขตของช่องหน้าต่าง
  บิต 1 จะถูกกำหนดให้เป็นค่า 0
- ถ้า xw__max-x < 0 แสดงว่าจุดพิกัดอยู่ทางขวาของช่องหน้าต่าง
  บิต 2 จะถูกกำหนดให้เป็นค่า 1
  ถ้า xw__max-x >= 0 แสดงว่าจุดพิกัดอยู่ภายในหรือบนขอบเขตของช่องหน้าต่าง
  บิต 2 จะถูกกำหนดให้เป็นค่า 0
- ถ้า yw__max-y < 0 แสดงว่าจุดพิกัดอยู่ข้างล่างช่องหน้าต่าง
  บิต 3 จะถูกกำหนดให้เป็นค่า 1
  ถ้า yw__max-y >= 0 แสดงว่าจุดพิกัดอยู่ภายในหรือบนขอบเขตของช่องหน้าต่าง
  บิต 3 จะถูกกำหนดให้เป็นค่า 0
- ถ้า y-yw__min < 0 แสดงว่าจุดพิกัดอยู่ข้างบนช่องหน้าต่าง
  บิต 4 จะถูกกำหนดให้เป็นค่า 1
  ถ้า y-yw__min >= 0 แสดงว่าจุดพิกัดอยู่ภายในหรือบนขอบเขตของช่องหน้าต่าง
  บิต 4 จะถูกกำหนดให้เป็นค่า 0

หมายเหตุ เราจะใช้ระบบพิกัดของจอภาพเป็นระบบอ้างอิง (จุดกำเนิด (0,0) อยู่ที่มุมบนซ้าย)

เมื่อเราสามารถกำหนดรหัสให้กับจุดพิกัดปลายของทุก ๆ จุดแล้ว เราจะสามารถตรวจสอบได้ว่าเส้นใดบ้างที่อยู่ภายในช่องหน้าต่างทั้งเส้นหรือภายนอกช่องหน้าต่างทั้งเส้น โดยดูจากรหัส ถ้าจุดพิกัดปลายของเส้นนั้นมีรหัสเป็น 0000 ทั้งคู่ (เส้นหนึ่งเส้นมีจุดพิกัดปลาย 2 จุด) แสดงว่าเส้นเส้นนั้นอยู่ภายในช่องหน้าต่างแน่นอน และถ้ารหัสของจุดพิกัดทั้งคู่มีค่าของบิตที่ตำแหน่งตรงกันเป็น 1 เช่น 1001 กับ 1010 จะเห็นว่าบิต 4 เป็น 1 เหมือนกัน แสดงว่าเส้นเส้นนี้อยู่ภายนอกช่องหน้าต่างแน่นอนซึ่งเราจะขริบออกไปได้เลย วิธีการตรวจสอบง่าย ๆ ทำได้โดยใช้การ AND กัน กล่าวคือ นำรหัสของจุดพิกัดทั้งคู่ของเส้นนั้นมาทำการ AND กัน ถ้าผลที่ได้ไม่เป็น 0000 ก็สรุปได้ทันทีว่าเส้นเส้นนั้นอยู่ภายนอกช่องหน้าต่าง

มีเส้นบางเส้นที่ไม่สามารถบอกได้ทันทีว่า อยู่ภายในหรือภายนอกช่องหน้าต่าง เช่น เส้นที่ปรากฏในรูปที่ 3.11 (เส้น P_3P_4 และ P_1P_2) การตรวจสอบเส้นเหล่านี้จะอาศัยการตัดกันของขอบเขตของช่องหน้าต่างกับเส้น ซึ่งมีวิธีการดังต่อไปนี้

เราจะอธิบายวิธีการโดยอ้างอิงเส้นในรูปที่ 3.11 เริ่มจากเส้น P_1P_2 จะตรวจสอบก่อนว่าจุด P_1 อยู่ตรงพื้นที่ส่วนใด โดยเทียบค่าพิกัดกับขอบเขต ซ้าย ขวา ล่าง และบน ตามลำดับ ซึ่งในที่นี้จะพบว่าจุด P_1 อยู่ข้างล่างช่องหน้าต่าง จากนั้นก็หาจุดตัดของเส้นกับขอบเขตล่างซึ่งให้เป็นจุด P_1' เส้น P_1P_1' จะถูกตัดออกไป เหลือเพียงเส้น $P_1' P_2$ ต่อไปก็ตรวจสอบว่า จุด P_2 อยู่ตรงพื้นที่ส่วนใด พบว่าอยู่ข้างบนของช่องหน้าต่าง คำนวณหาจุดตัดกับเส้นขอบเขตบนได้จุด P_2' เส้น $P_1' P_2'$ จะถูกเก็บไว้เป็นภาพเส้นที่เหลือหลังจากการขริบแล้ว สำหรับเส้น P_3P_4 ก็จัดการได้โดยวิธีเดียวกัน ตรวจพบว่าจุด P_3 อยู่ทางซ้ายของช่องหน้าต่าง หาจุดตัดกับเส้นขอบเขตซ้ายได้จุด P_3' เส้น P_3P_3' ถูกตัดออกไป เหลือเพียงเส้น $P_3' P_4$ ซึ่งเมื่อนำไปตรวจสอบต่อก็จะพบว่าอยู่ข้างล่างช่องหน้าต่างจึงขริบออกได้

สำหรับการคำนวณหาจุดตัดของเส้นกับเส้นขอบเขตหาได้จากสมการเส้นตรง เส้นตรงที่มีจุดพิกัดปลายเป็น (x1,y1) และ (x2,y2) ค่าพิกัด y ของจุดตัดสามารถคำนวณได้จากสมการ

$$y = y1 + m(x-x1)$$

โดยที่ x อาจจะกำหนดให้เป็น xw__min หรือ xw__max ก็ได้ ค่าความลาดชัน m หาได้จากสมการ

$$m = \frac{(y2-y1)}{(x2-x1)}$$

ในทำนองเดียวกัน ถ้าเราต้องการหาค่าพิกัด x ของจุดตัด ก็สามารถหาได้จากสมการ

$$x = x1 + \frac{(y-y1)}{m}$$

โดยที่ y อาจจะกำหนดให้เป็น yw__min หรือ yw__max ก็ได้

โปรแกรมที่ 3.2 เป็นโปรแกรมแสดงวิธีโคเฮน-ซูเธอร์แลนด์ ในการขริบเส้น

```
/* A C program that demonstrates
 * Cohen-Sutherland Clipping algorithm
*/

#include <graphics.h>

#define FOUR 4
#define XW_MIN 200
#define XW_MAX 400
#define YW_MIN 100
#define YW_MAX 300

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line(int x1,int y1,int x2,int y2);
void Out_Text_XY(int x,int y,char text[]);
void Set_Viewport(int lt,int tp,int rt,int bt);
void Clear_Viewport(void);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
void Encode(int x,int y,int c[FOUR]);
int Accept(int c1[FOUR],int c2[FOUR]);
int Reject(int c1[FOUR],int c2[FOUR]);
void Swap(int *x1,int *y1,int *x2,int *y2);

main()
{
    int code1[FOUR],code2[FOUR];
    int done,display;
    float m;
    int x1 = 250, x2 = 300, y1 = 50, y2 = 350;
```

```
Initialize_Graphics_Mode();

/* draw the frame of a window */
Draw_Rectangle(XW_MIN,YW_MIN,XW_MAX,YW_MAX);
/* draw a line across the window */
Draw_Line(x1,y1,x2,y2);
Out_Text_XY(150,10,"A line acroos the window without clipping");
getchar();
Clear_Viewport();

/* clipping algorithm */
done = 0; display = 0;
while (!done) {
        Encode(x1,y1,code1);
        Encode(x2,y2,code2);
        if(Accept(code1,code2)) { /* completely inside the window */
             done = 1; display = 1;
        }
        else { /* completely outside the window */
             if(Reject(code1,code2)) {
                  done = 1;
             }
             else { /* find intersection */
                  /* make sure that (x1,y1) is outside window */
                  if((code1[0]==0) && (code1[1]==0) &&
                     (code1[2]==0) && (code1[3]==0)) {
                          Swap(&x1,&y1,&x2,&y2);
          }
          if(x1!=x2) m = (y2-y1)/(x2-x1);
          if(code1[0]) { /* crosses left side */
               y1 = y1 + (XW_MIN-x1) * m;
               x1 = XW_MIN;
          }
          else  /* crosses right side */
```

```
            if(code1[1]) {
                y1 = y1 + (XW_MAX-x1) * m;
                x1 = XW_MAX;
            }
            else  /* crosses bottom side */
                if(code1[2]) {
                    if(x1==x2) x1 = x1;
                    else x1 = x1 + (YW_MAX-y1) / m;
                    y1 = YW_MAX;
                }
                else  /* crosses top side */
                    if(code1[3]) {
                        if(x1==x2) x1 = x1;
                        else x1 = x1 + (YW_MIN-y1) / m;
                        y1 = YW_MIN;
                    }
        } /* else Reject */
    }  /* else Accept */
} /* while */
if(display) {
        Draw_Rectangle(XW_MIN,YW_MIN,XW_MAX,YW_MAX);
        Draw_Line(x1,y1,x2,y2);
        Out_Text_XY(150,10,
            "A line acroos the window after clipped");
}
else {
        Draw_Rectangle(XW_MIN,YW_MIN,XW_MAX,YW_MAX);
        Out_Text_XY(150,10,
            "A line acroos the window after clipped");
  }
  getchar();
  Close_Graphics_Mode();
```

```
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
          printf("Graphics error : %s\n", grapherrormsg(errorcode));
          printf("Press any key to halt : ");
          getch();
          exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}
```

```
void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}

void Set_Viewport(int lt,int tp,int rt,int bt)
{
    setviewport(lt,tp,rt,bt,1);
}

void Clear_Viewport(void)
{
    clearviewport();
}

void Encode(int x,int y,int c[FOUR])
{     /* use screen coordinate system */
    if(x < XW_MIN) c[0] = 1;   /* left  of  window */
    else c[0] = 0;
    if(XW_MAX < x) c[1] = 1;   /* right of  window */
    else c[1] = 0;
    if(YW_MAX < y) c[2] = 1;   /* below the window */
    else c[2] = 0;
    if(y < YW_MIN) c[3] = 1;   /* above the window */
    else c[3] = 0;
}

int Accept(int c1[FOUR],int c2[FOUR])
```

```
{
     int k,a;

     /* if either point has value '1' in its code,
        thus, the line isn't completely inside the window */
     a = 1;
     for(k=0; k<4; k++)
            if(c1[k] || c2[k]) a = 0;
     return(a);
}

int Reject(int c1[FOUR],int c2[FOUR])
{
     int k,r;

     r = 0;
     for(k=0; k<4; k++) {
           if(c1[k] && c2[k]) r = 1;
     }
     return(r);
}

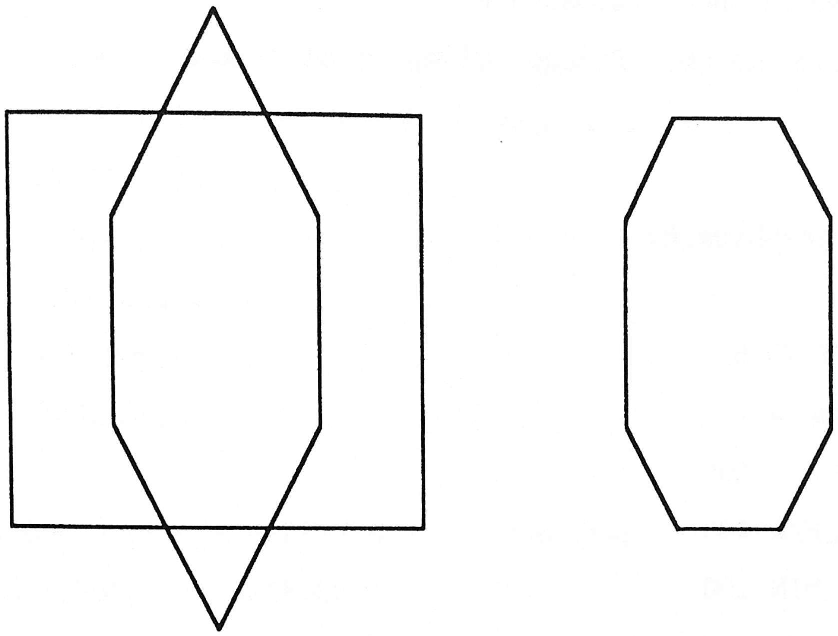
void Swap(int *x1,int *y1,int *x2,int *y2)
{
     int *temp;

     *temp = *x2; *x2 = *x1; *x1 = *temp;
     *temp = *y2; *y2 = *y1; *y1 = *temp;
}
```

โปรแกรมที่กล่าวมาข้างต้นยังมีส่วนผิดพลาดในบางกรณี ซึ่งจะเหลือไว้ให้ผู้อ่านไปปรับปรุงให้สมบูรณ์ยิ่งขึ้น

3.5.4 การขริบรูปเหลี่ยม

วิธีการตรวจสอบสำหรับการขริบเส้นไม่สามารถใช้กับการขริบรูปเหลี่ยมได้ เนื่องจากว่าในการขริบรูปเหลี่ยมซึ่งเป็นรูปปิด ภาพที่ได้ควรเป็นรูปปิดเช่นเดิม เพื่อให้ยังคงสามารถระบายสีหรือลวดลายได้ รูปที่ 3.12 แสดงให้เห็นว่าภาพที่ได้จากการขริบรูปเหลี่ยมยังคงเป็นรูปปิด วิธีที่ใช้ในการขริบก็มีด้วยกันหลายวิธี เช่น วิธีซูเธอร์แลนด์-ฮอดก์แมน (Sutherland-Hodgman Polygon Clipping Algorithm) เป็นต้น



รูปที่ 3.12 การขริบรูปเหลี่ยม

3.5.5 วิธีซูเธอร์แลนด์-ฮอดก์แมน

ภาพรูปเหลี่ยมสามารถนิยามได้ด้วยลำดับของจุดพิกัดชุดหนึ่ง ซึ่งจุดพิกัดเหล่านี้อาจจะอยู่ภายในช่องหน้าต่างหรืออยู่ภายนอกช่องหน้าต่างก็ได้ แต่หลังจากผ่านวิธีการขริบรูปเหลี่ยมแล้ว ผลที่ได้จะเป็นลำดับของจุดพิกัดอีกชุดหนึ่งซึ่งมีค่าอยู่ภายในขอบเขตของช่องหน้าต่างเท่านั้น

วิธีซูเธอร์แลนด์-ฮอดก์แมนมีขั้นตอนการทำงานดังนี้ เริ่มจากเปรียบเทียบพิกัดของจุดทีละจุดในลำดับกับพิกัดของเส้นขอบเขตของหน้าต่างทีละขอบเขต ผลที่ได้จะเป็นชุดของจุดพิกัดซึ่งจะนำไปใช้ในการวาดรูปเหลี่ยมซึ่งถูกขริบไปแล้ว นั่นคือจุดพิกัดที่ได้จะมีค่าอยู่ภายในขอบเขตของช่องหน้าต่างเท่านั้น สำหรับโปรแกรมตัวอย่างข้างล่างนี้จะนำโปรแกรมที่ 3.2 มาประยุกต์ใช้กับการขลิบรูปเหลี่ยม โดยจะตรวจสอบจุดพิกัดปลายของเส้นซึ่งประกอบเป็นรูปเหลี่ยมทีละคู่ ว่าอยู่ภายในช่องหน้าต่าง ภายนอกช่องหน้าต่างหรือจุดหนึ่งอยู่ภายในอีกจุดอยู่ภายนอก ถ้าจุดทั้งสองอยู่ภายใน

ช่องหน้าต่างก็ให้เก็บค่าจุดแรกไว้ก่อน (เพื่อนำไปเขียนรูปเหลี่ยมต่อไป) ถ้าจุดอยู่ภายนอกทั้งคู่ก็จะไม่เก็บเลย ส่วนกรณีที่จุดหนึ่งอยู่ภายในอีกจุดอยู่ภายนอก ก็จะหาจุดตัดระหว่างเส้นขอบของรูปเหลี่ยมกับเส้นขอบเขตของช่องหน้าต่าง แล้วเก็บค่าจุดตัดไว้ ทำเช่นนี้ไปจนได้จุดพิกัดซึ่งนิยามรูปเหลี่ยมที่ถูกขริบแล้ว

โปรแกรมที่ 3.3 แสดงวิธีซูเธอร์แลนด์-ฮอดก์แมน ในการขริบรูปเหลี่ยม

```
/* A C program that demonstrates
 * Sutherland-Hodgman Polygon Clipping algorithm
*/

#include <graphics.h>

#define PTS_NO 5
#define FOUR 4
#define XW_MIN 200
#define XW_MAX 400
#define YW_MIN 100
#define YW_MAX 300

typedef struct coord {  /* structure of points which define image */
        int x;
        int y;
        int c;
};

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Out_Text_XY(int x,int y,char text[]);
void Set_Viewport(int lt,int tp,int rt,int bt);
void Clear_Viewport(void);
```

```
void Draw_Rectangle(int lt,int tp,int rt,int bt);
void Draw_Image(struct coord imagedef[PTS_NO]);
void Encode(int x,int y,int c[FOUR]);
int Inside(int c1[FOUR],int c2[FOUR]);
int Outside(int c1[FOUR],int c2[FOUR]);
void Swap(int *x1,int *y1,int *x2,int *y2,int c1[FOUR],int c2[FOUR]);
main()
{
    /* structure of points which define image  */
    struct coord image[PTS_NO] = { /* image definition */
            250,150,1,
            300,50,1,
            350,75,1,
            375,200,1,
            250,150,1
    };
    struct coord new_image[PTS_NO]; /* new image after clipped */
    int code1[FOUR],code2[FOUR];
    int x1,y1,x2,y2,done,i,j;
    float m;

    Initialize_Graphics_Mode();

    /* draw the frame of a window */
    Draw_Rectangle(XW_MIN,YW_MIN,XW_MAX,YW_MAX);
    /* draw original image */
    Draw_Image(image);
    Out_Text_XY(265,25,"original image");
    Out_Text_XY(150,10,"A polygon and the window without clipping");
    getchar();
    Clear_Viewport();

    /* clipping algorithm */
```

```
done = 0; i = 0; j = 0;
while (!done) {
        Encode(image[i].x,image[i].y,code1);
        Encode(image[i+1].x,image[i+1].y,code2);
        if(Inside(code1,code2)) { /* 2 points are inside window */
            /* save the points */
            new_image[j].x = image[i].x;
            new_image[j].y = image[i].y;
            j++;
        }
        else { /* 2 points are outside window */
            if(Outside(code1,code2)) {
                /* do nothing */
            }
            else { /* 1 point inside, 1 point outside */
                /* find intersection */
                /* make sure that first point is outside window */
                x1 = image[i].x; y1 = image[i].y;
                x2 = image[i+1].x; y2 = image[i+1].y;
                if((code1[0]==0) && (code1[1]==0) &&
                   (code1[2]==0) && (code1[3]==0)) {
                        new_image[j].x = x1;
                        new_image[j].y = y1;
                        j++;
                        Swap(&x1,&y1,&x2,&y2,code1,code2);
                }
                if(x1!=x2)
                    m = (y2-y1)/(x2-x1); /* slope */
                if(code1[0]) { /* crosses left side */
                    y1 = y1+(XW_MIN-x1)*m;
                    x1 = XW_MIN;
                    new_image[j].x = x1;
```

```
                        new_image[j].y = y1;
                        j++;
}
else  /* crosses right side */
    if(code1[1])'{
        y1 = y1+(XW_MAX-x1)*m;
        x1 = XW_MAX;
        new_image[j].x = x1;
        new_image[j].y = y1;
        j++;
    }
    else  /* crosses bottom side */
  if(code1[2]) {
    if(x1==x2) x1 = x1;
    else x1 = x1+(YW_MAX-y1)/m;
          y1 = YW_MAX;
          new_image[j].x = x1;
          new_image[j].y = y1;
          j++;
  }
  else  /* crosses top side */
    if(code1[3]) {
  if(x1==x2) x1 = x1;
  else x1 = x1+(YW_MIN-y1)/m;
       y1 = YW_MIN;
       new_image[j].x = x1;
       new_image[j].y = y1;
       j++;
     }
} /* else Outside */
}  /* else Inside */
if((i==(PTS_NO-1)) || (j==(PTS_NO))) done = 1;else i++;
```

```
    } /* while */
    Draw_Rectangle(XW_MIN,YW_MIN,XW_MAX,YW_MAX);
    Draw_Image(new_image);
    Out_Text_XY(265,25,"clipped image");
    Out_Text_XY(150,10,"A polygon and the window after clipping");

    getchar();
    Close_Graphics_Mode(); /* function in program 2.1 */
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
```

```
    closegraph();
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}

void Set_Viewport(int lt,int tp,int rt,int bt)
{
    setviewport(lt,tp,rt,bt,1);
}

void Clear_Viewport(void)
{
    clearviewport();
}
```

```
void Draw_Image(struct coord imagedef[PTS_NO])
{
    int i;

    for(i=0; i<PTS_NO ; i++) {
          if(i==0) {
              Move_To(imagedef[i].x,imagedef[i].y);
          }
          Draw_Line_To(imagedef[i].x,imagedef[i].y);
    }
}

void Encode(int x,int y,int c[FOUR])
{     /* use screen coordinate system */
    if(x < XW_MIN) c[0] = 1;    /* left  of  window */
    else c[0] = 0;
    if(XW_MAX < x) c[1] = 1;    /* right of  window */
    else c[1] = 0;
    if(YW_MAX < y) c[2] = 1;    /* below the window */
    else c[2] = 0;
    if(y < YW_MIN) c[3] = 1;    /* above the window */
    else c[3] = 0;
}

int Inside(int c1[FOUR],int c2[FOUR])
{
    int k,a;

    a = 1;
    for(k=0; k<4; k++)
         if(c1[k] || c2[k]) a = 0;
    return(a);
```

```
}

int Outside(int c1[FOUR],int c2[FOUR])
{
    int k,r;

    r = 0;
    for(k=0; k<4; k++)
        if(c1[k] && c2[k]) r = 1;
    return(r);
}

void Swap(int *x1,int *y1,int *x2,int *y2,int c1[FOUR],int c2[FOUR])
{
    int *temp,t[FOUR],k;

    *temp = *x2; *x2 = *x1; *x1 = *temp;
    *temp = *y2; *y2 = *y1; *y1 = *temp;
    for(k=0; k<FOUR; k++) {
        t[k] = c1[k]; c1[k] = c2[k]; c2[k] = t[k];
    }
}
```

โปรแกรมข้างต้นจะใช้ได้ในกรณีที่จุดเริ่มต้นอยู่ภายในช่องหน้าต่างทั้งคู่ หรืออยู่ภายในเพียงจุดเดียว แต่ถ้าจุดเริ่มต้นทั้งสองอยู่ภายนอกช่องหน้าต่างแล้วจะใช้ไม่ได้ ซึ่งจะปล่อยไว้ให้ผู้อ่านแก้ปัญหานี้เป็นแบบฝึกหัด

3.5.6 การขริบตัวอักษร

มีวิธีการหลายอย่างที่ใช้ในการขริบตัวอักษรที่แสดงในระบบกราฟิกส์ การจะเลือกใช้วิธีการใดนั้น ขึ้นอยู่กับวิธีการสร้างตัวอักษรและความซับซ้อนในการจัดการตัวอักษรที่ระบบต้องการ

วิธีการที่ง่ายที่สุดในการจัดการกับตัวอักษรที่แสดงในระบบกราฟิกส์ก็คือ จะใช้ขอบเขตของช่องหน้าต่างหรือช่องแสดงภาพเป็นตัวเปรียบเทียบ กล่าวคือถ้าตัวอักษรหรือข้อความที่แสดง

นั้นอยู่ภายในขอบเขตของช่องหน้าต่างทั้งหมดก็จะแสดงข้อความนั้น แต่ถ้ามีบางส่วนของข้อความถูกตัดโดยขอบเขตก็จะไม่แสดงข้อความนั้นเลย แม้ว่าจะมีบางส่วนอยู่ภายในช่องหน้าต่างก็ตาม

สำหรับวิธีการอื่น ๆ ที่ทำได้ยากขึ้นก็คือจะพิจารณาเป็นตัวอักษร นั่นคือถ้าอักษรตัวใดอยู่ภายนอกขอบเขตก็ให้ขริบออก ถ้าอยู่ภายในขอบเขตก็ให้แสดง สำหรับตัวอักษรบางตัวที่ถูกตัดโดยขอบเขตพอดี ก็จะเลือกว่าให้แสดงหรือขริบทิ้งก็ได้

วิธีการสุดท้ายที่ทำกันนั้นจะถือว่าตัวอักษรเป็นเหมือนเส้น ส่วนใดอยู่ภายนอกขอบเขตก็จะถูกขริบ ส่วนที่อยู่ภายในไม่ว่าจะมีครบรูปแบบตัวอักษรหรือไม่ก็ตามจะแสดงบนจอภาพ

3.6 ข้อความในระบบกราฟิกส์

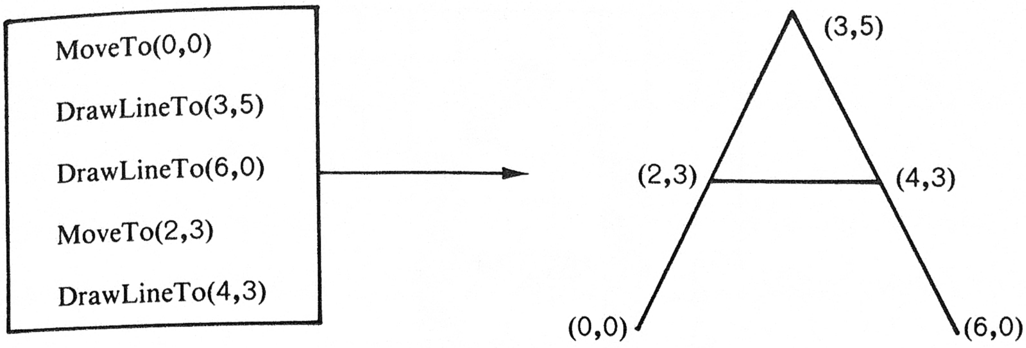
ภาพที่เราแสดงบนจอภาพส่วนใหญ่จะประกอบไปด้วยจุดและเส้น แต่ก็มีการประยุกต์ใช้งานบางอย่างซึ่งต้องการให้มีข้อความประกอบกับภาพที่แสดงด้วย เช่น ภาพกราฟในงานทางธุรกิจมักจะมีข้อความหรือตัวเลขประกอบเสมอ

รูปแบบแบบหนึ่งของชุดตัวอักษร เรียกว่า *ฟอนต์หรือแบบตัวอักษร (font or typeface)* ซึ่งในระบบกราฟิกส์มีวิธีการสร้างตัวอักษรได้ 2 วิธี คือ ตัวอักษรแบบเวกเตอร์ และตัวอักษรแบบแรสเตอร์

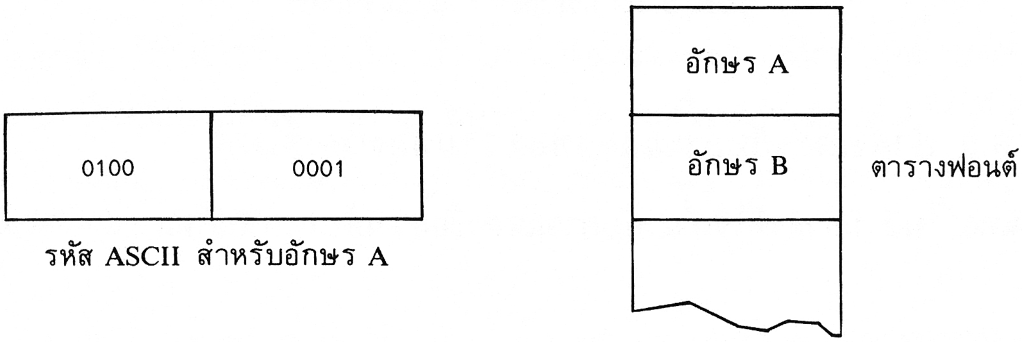
3.6.1 ตัวอักษรแบบเวกเตอร์

อักษรแบบนี้จะสร้างจากการนำเอาส่วนของเส้นตรงหลาย ๆ เส้นมาประกอบกันให้เป็นรูปตัวอักษรที่ต้องการ ซึ่งก็จะเหมือนกับวิธีการเขียนตัวอักษรของคนเรานั่นเอง (ดูรูปที่ 3.13) ตัวอักษรแต่ละตัวจะถูกเก็บไว้ในที่ที่เรียกว่า ตารางแบบตัวอักษร (font table) ซึ่งจะจัดเก็บไว้ในแผ่นดิสก์หรือในหน่วยความจำของคอมพิวเตอร์ก็ได้ ตัวอักษรแต่ละตัวจะจัดเก็บในรูปของคำสั่งเกี่ยวกับการสร้างเส้นและการวางตำแหน่ง การพิมพ์ตัวอักษรเข้าในโปรแกรม จะใช้รหัส ASCII เป็นตัวชี้ไปยังตารางแบบตัวอักษร ซึ่งทำให้เวลาเรากดอักษรใด อักษรนั้นก็จะปรากฏบนจอภาพ (ดูรูปที่ 3.14)

ข้อดีของตัวอักษรแบบเวกเตอร์ก็คือ สามารถจะแสดงตัวอักษรโดยให้มีขนาดเท่าไรก็ได้และจะเขียนในมุมใดก็ได้ กล่าวคือสามารถพิมพ์เอียงเป็นมุม พิมพ์ตั้งฉาก หรือพิมพ์ตีลังกาก็ได้ มีข้อเสียเล็กน้อยก็คือ การออกแบบตัวอักษรทำได้ค่อนข้างยาก



รูปที่ 3.13 ตัวอักษรแบบเวกเตอร์



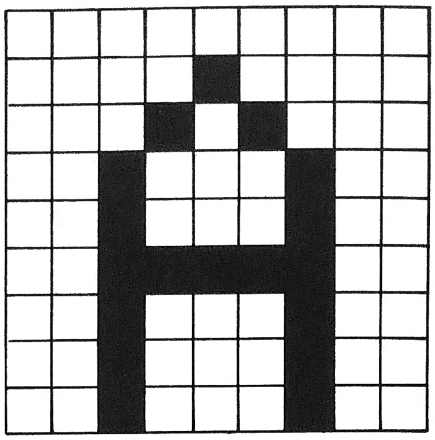
รูปที่ 3.14 การใช้รหัส ASCII เป็นตัวชี้ไปยังตารางแบบตัวอักษร

3.6.2 ตัวอักษรแบบแรสเตอร์

เป็นวิธีการสร้างตัวอักษรอีกแบบหนึ่งที่สร้างได้ง่ายกว่าแบบเวกเตอร์ กล่าวคือตัวอักษรทุกตัวจะสร้างจากตารางสี่เหลี่ยม ซึ่งเราจะกำหนดจุดต่อ ๆ กันเป็นรูปตัวอักษร (รูปที่ 3.15) ขนาดของตัวอักษรก็จะแตกต่างกันไปในระบบกราฟิกส์แต่ละระบบ เช่น เป็นตัวอักษรขนาด 10 × 20 จุดซึ่งก็คือ ใช้ตารางขนาด 10 × 20 จุด เป็นที่สำหรับออกแบบตัวอักษร เป็นต้น

ข้อดีของตัวอักษรแบบแรสเตอร์คือ สามารถออกแบบได้ง่าย ส่วนข้อเสียก็คือ การขยายขนาดจะทำให้รูปแบบตัวอักษรผิดเพี้ยนไปจากเดิม และการพิมพ์เอียงเป็นมุมต่าง ๆ ก็ทำได้ยาก

ชุดของแบบตัวอักษรอาจจะเก็บไว้ในหน่วยความจำประเภท ROM (read only memory) ของคอมพิวเตอร์หรือเก็บในแผ่นดิสก์ก็ได้ ในการออกแบบตัวอักษรแบบแรสเตอร์มักจะใช้โปรแกรมที่เรียกว่า *ฟอนต์อิดิเตอร์ (font editor)* เป็นตัวช่วยในการออกแบบ ซึ่งจะได้อธิบายรายละเอียดต่อไปในภายหลัง



รูปที่ 3.15 ตัวอักษร A แบบแรสเตอร์

3.6.3 คำสั่งเกี่ยวกับการแสดงข้อความของเทอร์โบซี

ในเทอร์โบซี มีคำสั่งที่ใช้เกี่ยวกับการแสดงข้อความในกราฟิกส์หลายคำสั่งดังต่อไปนี้

gettextsettings	ใช้สำหรับตรวจสอบว่า ฟอนต์ตัวอักษรที่ใช้ในกราฟิกส์ขณะนั้นเป็นอย่างไร เช่น เป็นฟอนต์แบบใด ขนาดเท่าไร เป็นต้น
outtext	ใช้สำหรับแสดงตัวอักษรหรือข้อความตรงจุด CP ในช่องแสดงภาพ
outtextxy	ใช้สำหรับแสดงตัวอักษรหรือข้อความตรงตำแหน่งที่กำหนด
settextjustify	ใช้สำหรับแสดงตัวอักษรหรือข้อความ โดยข้อความที่แสดงจะสัมพันธ์กับจุด CP เช่น ข้อความที่แสดงจะมีจุด CP อยู่ตรงกลาง เป็นต้น
settextstyle	ใช้สำหรับกำหนดลักษณะของตัวอักษรหรือข้อความ และกำหนดให้แสดงข้อความในแนวดิ่งหรือแนวนอน
setusercharsize	ใช้สำหรับกำหนดขนาดของตัวอักษรแบบเวกเตอร์ หรือ stroked fonts
textheight	ใช้สำหรับแสดงค่าส่วนสูงของตัวอักษรในหน่วยพิกเซล
textwidth	ใช้สำหรับแสดงค่าความกว้างของตัวอักษรในหน่วยพิกเซล

สำหรับรายละเอียดของแต่ละคำสั่งข้างต้นและตัวอย่างการใช้งานสามารถศึกษาได้จากคู่มือเทอร์โบซี

3.7 สี

สีในระบบกราฟิกส์ทำให้ภาพที่สร้างได้สมจริงมากขึ้น น่าสนใจมากขึ้น แต่จอภาพที่ใช้งานจะต้องสามารถแสดงสีได้ ซึ่งจอภาพที่แสดงสีได้จะมีราคาแพงกว่าจอภาพแบบโมโนโครมมาก และจอภาพที่แสดงสีได้มากก็มักจะมีราคาแพงกว่าจอภาพที่แสดงสีได้น้อย สำหรับในหนังสือเล่มนี้เราจะไม่สนใจกับการใช้สี เนื่องจากต้องการเสนอเรื่องทั่ว ๆ ไปเกี่ยวกับคอมพิวเตอร์กราฟิกส์เท่านั้น

3.8 สรุป

ในบทนี้เราได้กล่าวถึงเรื่องที่ควรรู้เกี่ยวกับคอมพิวเตอร์กราฟิกส์ เช่น ช่องหน้าต่างกับช่องแสดงภาพ การขริบแบบต่าง ๆ และตัวอักษรหรือข้อความที่แสดงในระบบกราฟิกส์ เรื่องเหล่านี้เราควรทำความเข้าใจไว้เพื่อจะได้เป็นพื้นฐานในการทำความเข้าใจเรื่องอื่น ๆ ต่อไป

4 แอตทริบิวต์ของคำสั่งกราฟิกส์

4.1 บทนำ

การแสดงภาพพื้นฐานในระบบกราฟิกส์นั้นอาจจะแสดงได้หลาย ๆ แบบ โดยใช้คำสั่งการสร้างภาพคำสั่งเดียวกันแต่เปลี่ยนลักษณะเฉพาะหรือแอตทริบิวต์ (attributes) ของคำสั่ง เช่น เส้นอาจจะเป็นเส้นทึบ เส้นบาง เส้นหนา เส้นประ หรือเป็นเส้นขีด ๆ ต่อกัน สำหรับรูปปิดก็อาจจะเป็นรูปปิดที่ภายในพื้นที่ว่าง ๆ พื้นที่ถูกระบายด้วยสีใดสีหนึ่ง หรือถูกระบายด้วยลวดลายก็ได้ ส่วนตัวอักษรหรือข้อความที่แสดงบนจอภาพก็อาจแสดงจากซ้ายไปขวา บนลงล่าง หรือในแนวเฉียง และตัวอักษรแต่ละตัวอาจจะมีสีและขนาดต่าง ๆ กันไปได้ด้วย

วิธีการหนึ่งที่ใช้ในการกำหนดแอตทริบิวต์ให้กับคำสั่งคือ เพิ่มแอตทริบิวต์เข้าไปในรายการพารามิเตอร์ของคำสั่ง ตัวอย่างเช่น คำสั่งสร้างจุดมีพารามิเตอร์เป็นค่า x กับค่า y ถ้าเราต้องการกำหนดสีของจุดได้ด้วย เราก็เพิ่มพารามิเตอร์ color เข้าไปในคำสั่งสร้างจุดอีก 1 ตัว เมื่อใช้คำสั่งนี้เราก็สามารถสร้างจุดซึ่งมีสีและตำแหน่งตามต้องการได้

อีกวิธีหนึ่งที่ใช้ในการกำหนดแอตทริบิวต์ก็คือ ให้การกำหนดแอตทริบิวต์เป็นคำสั่ง 1 คำสั่งเลย การใช้คำสั่งนี้ก็จะเป็นการไปกำหนดแอตทริบิวต์นั้นไว้ก่อน เมื่อเราใช้คำสั่งการสร้างภาพกราฟิกส์ ภาพก็จะถูกสร้างตามแอตทริบิวต์ที่กำหนดไว้ครั้งหลังสุด และการกำหนดนี้จะคงอยู่ตลอดไปจนกว่าจะมีคำสั่งเปลี่ยนแอตทริบิวต์ใหม่ เช่น อาจจะมีคำสั่งกำหนดสีที่ใช้ในการแสดงภาพบนจอภาพ เมื่อใช้คำสั่งกำหนดแล้ว คำสั่งสร้างจุดก็จะสร้างจุดซึ่งมีสีตามที่กำหนด คำสั่งสร้างเส้นก็จะสร้างเส้นซึ่งมีสีตามที่กำหนดเช่นกัน การเปลี่ยนสีทำได้โดยใช้คำสั่งกำหนดสีใหม่อีกครั้งเป็นสีใหม่ที่เราต้องการ จากนั้นภาพที่สร้างต่อไปก็จะมีสีตามการกำหนดครั้งหลังนี้

4.2 สไตล์ของเส้น

แอตทริบิวต์ที่ใช้ร่วมกับคำสั่งการสร้างเส้นได้แก่ แบบของเส้น ความกว้างหรือความหนาของเส้น และสีของเส้น คำสั่งการสร้างเส้นอาจจะมีแอตทริบิวต์เหล่านี้เป็นพารามิเตอร์หรืออาจจะทำเป็นคำสั่งต่างหากเพื่อใช้กำหนดแอตทริบิวต์ก็ได้

4.2.1 แบบของเส้น

แอตทริบิวต์แบบของเส้นที่ใช้กันทั่วไปได้แก่ เส้นหนา เส้นประ และเส้นที่เป็นจุดต่อ ๆ กัน เส้นเหล่านี้สร้างได้โดยปรับปรุงจากวิธีการสร้างเส้นที่ได้กล่าวมาแล้ว เช่น เส้นประ ทำได้โดยการสร้างเส้นสั้น ๆ สลับกับการสร้างช่องว่างเล็ก ๆ ส่วนเส้นที่เป็นจุดต่อ ๆ กันก็ทำได้โดยการพล็อตจุดสลับกับไม่พล็อตจุดที่ตำแหน่งต่อมา หรืออาจจะพล็อตจุดทุก ๆ พิกเซลที่ 3 นับจากจุดที่พล็อตก็ได้ นอกจาก 3 แบบที่กล่าวมาแล้วก็อาจจะมีแบบอื่น ๆ ได้ขึ้นกับความต้องการของผู้ใช้นั่นเอง

4.2.2 ความกว้าง (ความหนา) ของเส้น

แอตทริบิวต์นี้จะใช้กำหนดความหนาของเส้นที่จะแสดงผล กรณีที่เป็นเส้นแบบแรสเตอร์ก็อาจจะกำหนดเป็นความหนากี่พิกเซล เช่น ความหนา 1 พิกเซล หรือความหนา 3 พิกเซล สำหรับเส้นแบบเวกเตอร์ ซึ่งสร้างความหนาได้โดยลากเส้นขนานกันโดยให้ระยะระหว่างเส้นน้อยมาก ๆ การกำหนดความหนาก็อาจจะใช้วิธีกำหนดจำนวนเส้นที่ลากขนานก็ได้ โดยทั่วไปการกำหนดความหนาของเส้นจะทำดังนี้คือ กำหนดโดยใช้เลขจำนวนเต็มบวก เช่น กำหนดให้เลข 1 เป็นความหนาของเส้นขนาดมาตรฐานอันหนึ่ง การกำหนดเลขที่มากกว่า 1 ก็จะให้เส้นที่มีขนาดหนากว่าเดิม ตัวอย่างเช่น กำหนดเป็น 2 จะหมายถึงเส้นมีขนาดความหนาเป็น 2 เท่าของเส้นขนาดมาตรฐาน หรือกำหนดเป็น 0.5 จะหมายถึงเส้นที่จะสร้างมีขนาดความหนาเป็นครึ่งหนึ่งของเส้นซึ่งมีความหนามาตรฐานเป็นต้น

4.2.3 สีของเส้น

คำสั่งในการสร้างเส้นมักจะไม่มีพารามิเตอร์สีเพิ่มเข้าไปในคำสั่ง เนื่องจากจะทำให้คำสั่งรุงรังเกินไป ดังนั้นจึงมักมีคำสั่งการกำหนดสีที่จะใช้ในการแสดงผลภาพ ซึ่งเมื่อใช้คำสั่งนี้กำหนดแล้วการแสดงผลภาพก็จะใช้สีดังกล่าวเสมอ จนกว่าจะมีการเปลี่ยนการกำหนดใหม่

4.2.4 การกำหนดแอตทริบิวต์ของเส้นในเทอร์โบซี

สำหรับในเทอร์โบซี การกำหนดแอตทริบิวต์แบบของเส้นและความกว้างหรือความหนาของเส้นทำได้โดยใช้คำสั่ง setlinestyle และการกำหนดสีทำได้โดยใช้คำสั่ง setcolor

เนื่องจากคำสั่งทั้งสองข้างต้นเป็นคำสั่งที่ใช้เฉพาะกับเทอร์โบซีเท่านั้น ดังนั้นเพื่อให้การเขียนโปรแกรมตัวอย่างในหนังสือเล่มนี้ไม่ผูกพันกับเทอร์โบซี เราจะกำหนดคำสั่งใหม่เป็น

Set__Line__Attributes(line__type, line__width, line__color)

ซึ่งเป็นคำสั่งสำหรับการกำหนดแบบของเส้น ความหนาและสีของเส้น

โปรแกรมที่ 4.1 ใช้คำสั่ง Set__Line__Attributes และแสดงผลเส้นหลังจากการกำหนดแอตทริบิวต์แล้ว

```
/* A C program that demonstrates how to set line attributes
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line(int x1,int y1,int x2,int y2);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
void Set_Line_Attributes(int line_type,int line_width,int line_color);

main()
{
    Initialize_Graphics_Mode();

    Set_Line_Attributes(0,1,2);
    Draw_Line(0,50,Get_Max_X(),50);

    Set_Line_Attributes(0,3,3);
    Draw_Line(0,100,Get_Max_X(),100);

    Set_Line_Attributes(1,1,4);
```

```
    Draw_Line(0,150,Get_Max_X(),150);

    Set_Line_Attributes(3,3,9);
    Draw_Line(0,200,Get_Max_X(),200);

    Set_Line_Attributes(2,1,14);
    Draw_Line(0,250,Get_Max_X(),250);

    Out_Text_XY(200,300,"Lines with various attributes");
    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}
```

```
void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

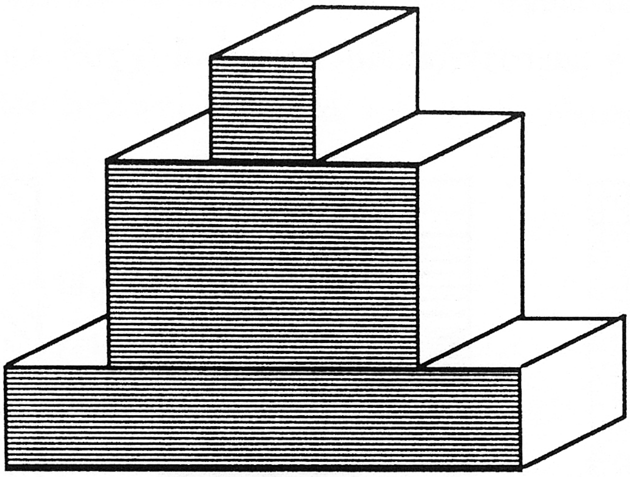
int Get_Max_X(void)
{
    return(getmaxx());
}

void Set_Line_Attributes(int line_type,int line_width,int line_color)
{
    setlinestyle(line_type,1,line_width);
    setcolor(line_color);
}
```

สำหรับพารามิเตอร์ที่ใช้ในคำสั่ง Set_Line_Attributes นั้น ซึ่งได้แก่ line_type,line_width และ line_color ค่าที่กำหนด ได้มาจากการกำหนดของคำสั่ง setlinestyle และ setcolor ในเทอร์โบซีนั่นเอง เช่น สีที่สามารถกำหนดได้จะมีค่าระหว่าง 0-15 เป็นต้น สำหรับรายละเอียดต่าง ๆ ของคำสั่งทั้งสองนี้สามารถศึกษาได้จากคู่มือเทอร์โบซีจะไม่กล่าวในที่นี้

4.3 การระบายลวดลายหรือสีให้กับรูปปิด

จุดประสงค์ของการระบายลวดลายหรือสีให้กับรูปปิดก็เพื่อให้ภาพที่ได้มีความสมจริงมากขึ้น จอภาพแบบแรสเตอร์ได้รับความนิยมสูงกว่าจอภาพแบบเวกเตอร์ก็เนื่องมาจากเป็นจอภาพที่สามารถทำการให้สีหรือลวดลายกับภาพได้ง่ายกว่าจอภาพแบบเวกเตอร์ รูปที่ 4.1 เป็นภาพที่แสดงการระบายสีและลวดลายซึ่งทำให้ภาพที่ได้น่าสนใจมากขึ้นและดูมีมิติความลึก

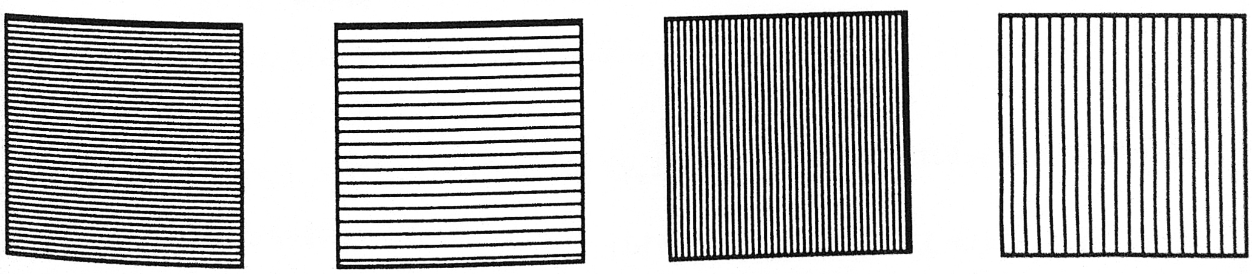


รูปที่ 4.1 ภาพที่มีการระบายลวดลายหรือสี

ในหัวข้อนี้เราจะมาศึกษาถึงเทคนิคเบื้องต้นในการระบายลวดลายหรือสี ซึ่งในหนังสือเล่มนี้จะใช้เพียงสีขาวและดำเท่านั้น แม้ว่าภาพที่ได้จากการใช้สีเพียง 2 สีจะมีคุณภาพต่ำแต่ก็สามารถทำให้ภาพที่ได้น่าสนใจมากกว่าภาพที่ไม่มีการระบายลวดลายหรือสีใด ๆ เลย

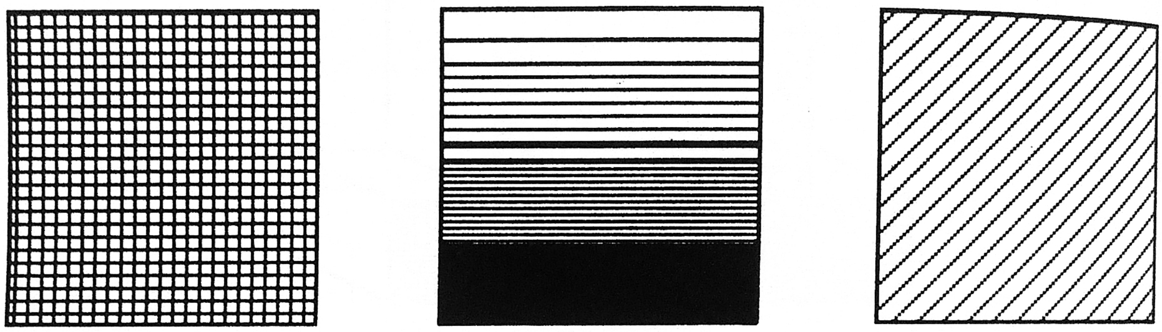
4.3.1 การระบายลวดลายในรูปสี่เหลี่ยม

การระบายลวดลายให้รูปสี่เหลี่ยมมีอยู่ 2 แบบคือ ระบายโดยใช้เส้นและระบายโดยใช้จุด



รูปที่ 4.2 การระบายลวดลายโดยใช้เส้นที่มีระยะห่างระหว่างเส้นแตกต่างกันไป

1. การระบายลวดลายโดยใช้เส้น วิธีระบายโดยใช้เส้นแบบที่ง่ายที่สุดก็คือ ใช้เส้นขนานในแนวนอนหรือแนวตั้งลากผ่านพื้นที่ของสี่เหลี่ยม การใช้เส้นขนานที่มีระยะห่างระหว่างเส้นต่างกันจะทำให้ได้ผลของการระบายต่างกันไป รูปที่ 4.2 แสดงผลของการระบายลวดลายโดยใช้เส้นที่มีระยะห่างระหว่างเส้นแตกต่างกันจะเห็นว่า ถ้าให้เส้นขนานที่มีระยะห่างระหว่างเส้นน้อยมาก ๆ หรือไม่มีเลย เราก็จะได้ภาพสี่เหลี่ยมที่มีการระบายสีแบบทึบนั่นเอง โปรแกรมที่ 4.2 เป็นโปรแกรมที่ใช้สำหรับระบายรูปสี่เหลี่ยมโดยใช้เส้นในแบบต่าง ๆ เช่นใช้เส้นขนานที่มีระยะห่างระหว่างเส้นค่าหนึ่ง ใช้เส้นขนานในแนวนอนกับเส้นขนานในแนวตั้งตัดกัน ใช้เส้นขนานซึ่งมีระยะห่างระหว่างเส้นเปลี่ยนแปลงเป็นขั้น ๆ และการใช้เส้นเฉียงระบายพื้นที่ (ดูรูปที่ 4.3)



(ก) (ข) (ค)

(ก) การใช้เส้นขนานในแนวนอนและแนวตั้งตัดกัน

(ข) การใช้ระยะห่างระหว่างเส้นขนานเปลี่ยนแปลงเป็นขั้น ๆ

(ค) การระบายเส้นขนานในแนวเฉียง

รูปที่ 4.3

โปรแกรมที่ 4.2

```
/* A C program that demonstrates various line shading
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line(int x1,int y1,int x2,int y2);
void Out_Text_XY(int x,int y,char text[]);
```

```
void Draw_Rectangle(int lt,int tp,int rt,int bt);

main()
{
    int y,factor;

    Initialize_Graphics_Mode();

    /* intersection between hor. and ver. line shading */
    Draw_Rectangle(10,10,110,110);
    for(y=15; y<110; y+=5) {
         Draw_Line(10,y,110,y);
         Draw_Line(y,10,y,110);
    }

    /* horizontal line shading */
    Draw_Rectangle(10,200,110,300);
    for(y=205; y<300; y+=5)
         Draw_Line(10,y,110,y);

    /* vary the space between the shaded line */
    Draw_Rectangle(200,10,300,110);
    for(y=10,factor=1; y<110,factor<14; y+=factor,factor++)
          Draw_Line(200,y,300,y);
    /* diagonal line shading */
    Draw_Rectangle(200,200,300,300);
    for(y=0; y<=100; y+=5)
          Draw_Line(200,200+y,200+y,200);
    for(y=0; y<=100; y+=5)
          Draw_Line(200+y,300,300,200+y);
    Out_Text_XY(100,350,"Various line shading");

    getchar();
```

```
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
          printf("Graphics error : %s\n", grapherrormsg(errorcode));
          printf("Press any key to halt : ");
          getch();
          exit(1); /* return with error code */
    }
}

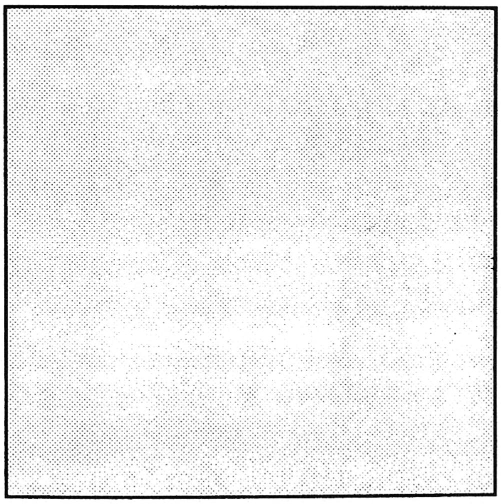
void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}
```

```
void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}
```

2. การระบายลวดลายโดยใช้จุด นอกจากการระบายลวดลายโดยใช้เส้นแล้ว การใช้จุดก็สามารถทำให้เกิดภาพได้อีกแบบหนึ่ง เช่น การพล็อตจุดโดยการสุ่มพิกัดภายในสี่เหลี่ยม (รูปที่ 4.4) จะได้ภาพสี่เหลี่ยมที่มีการระบายลวดลายในแบบที่แตกต่างจากแบบที่ใช้เส้นระบาย



รูปที่ 4.4 การระบายลวดลายโดยใช้จุด

โปรแกรมที่ 4.3 การระบายลวดลายโดยใช้จุดพล็อตลงในพื้นที่สี่เหลี่ยมแบบสุ่มค่าพิกัด

```
/* A C program that demonstrates point shading
*/

#include <graphics.h>
```

```
#include <stdlib.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Out_Text_XY(int x,int y,char text[]);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
void Randomize(void);
int Random(int num);

main()
{
    int i;

    Initialize_Graphics_Mode();

    Draw_Rectangle(10,10,110,110);
    Randomize();
    for(i=0; i<=100; i++)
        Plot_Point(10+Random(100),10+Random(100),15);

    Draw_Rectangle(150,10,250,110);
    Randomize();
    for(i=0; i<=500; i++)
        Plot_Point(150+Random(100),10+Random(100),15);

    Draw_Rectangle(10,150,110,250);
    Randomize();
    for(i=0; i<=1500; i++)
        Plot_Point(10+Random(100),150+Random(100),15);

    Draw_Rectangle(150,150,250,250);
```

```
    Randomize();
    for(i=0; i<=2000; i++)
          Plot_Point(150+Random(100),150+Random(100),15);

    Out_Text_XY(50,300,"various point shading");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
          printf("Graphics error : %s\n", grapherrormsg(errorcode));
          printf("Press any key to halt : ");
          getch();
          exit(1); /* return with error code */
    }
}
void Close_Graphics_Mode(void)
{
```

```
    closegraph();
}

void Plot_Point(int x,int y,int color)
{
    putpixel(x,y,color);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}

void Randomize(void)  /* initializes random number generator */
{
    randomize();
}

int Random(int num)  /* random number generator */
{
    return(random(num));
}
```

4.3.2 การระบายลวดลายในรูปวงกลม

เราสามารถใช้เส้นรัศมี เส้นโค้ง หรือจุด ระบายภายในรูปวงกลมได้ เช่น ถ้าใช้เส้นโค้ง เราทำได้โดยการวาดวงกลมซ้อน ๆ กันนั่นเอง ถ้าใช้จุด เราก็จะพล็อตจุดภายในพื้นที่ของวงกลม อาจจะสุ่มพิกัดหรือไม่ก็ได้ซึ่งภาพที่ได้ก็จะแตกต่างกันไป

โปรแกรมที่ 4.4 การระบายลวดลายในวงกลมโดยใช้ จุด เส้นโค้งหรือเส้นรอบวง และเส้นรัศมี

```
/* A C program that demonstrates circle shading
*/

#include <graphics.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.14159

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Out_Text_XY(int x,int y,char text[]);
void Draw_Line(int x1,int y1,int x2,int y2);
void Draw_Circle(int xc,int yc,int radius);
void Randomize(void);
int Random(int num);

main()
{
    int i,theta;
    int x,y;

    Initialize_Graphics_Mode();

    /* circumference shading */
    Draw_Circle(150,150,100);
    for(i=0; i<100; i+=5)
          Draw_Circle(150,150,i);

    /* radius shading */
```

```
    Draw_Circle(400,150,100);
    for(theta=0; theta<=360; theta+=5) {
        x = 400 + 100 * cos(theta*PI/180);
        y = 150 + 100 * sin(theta*PI/180);
        Draw_Line(400,150,x,y);
    }

    /* point shading */
    Draw_Circle(275,300,50);
    Randomize();
    for(theta=0; theta<=360; theta+=5) {
       x = 275 + 50 * cos(theta*PI/180);
       y = 300 + 50 * sin(theta*PI/180);
       for(i=0; i<=3; i++)
           Plot_Point(275+Random(x-275),300+Random(y-300),15);
    }

    Out_Text_XY(200,380,"various point shading");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
```

```
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Plot_Point(int x,int y,int color)
{
    putpixel(x,y,color);
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}

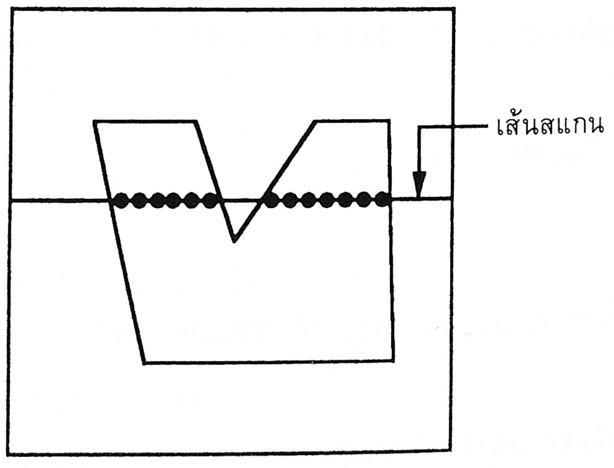
void Draw_Circle(int xc,int yc,int radius)
{
```

```
    circle(xc,yc,radius);
}
void Randomize(void)  /* initializes random number generator */
{
    randomize();
}

int Random(int num)  /* random number generator */
{
    return(random(num));
}
```

4.3.3 การระบายสีทึบลงในรูปปิด

การระบายสีทึบก็คือการทำให้รูปปิดนั้นมีสีใดสีหนึ่งโดยไม่มีลวดลายใด ๆ เลย เช่น เป็นรูปสี่เหลี่ยมที่ระบายด้วยสีดำ หรือ ขาว (หรือเป็นสีอื่น ๆ กรณีที่จอภาพแสดงสีได้มากกว่าสีขาว-ดำ) วิธีการระบายสีทึบนี้มีอยู่หลายวิธีแต่ละวิธีก็มีข้อดีข้อเสียแตกต่างกันไปบ้าง สำหรับหนังสือเล่มนี้จะอธิบายหลักการทำงานของวิธีต่าง ๆ เหล่านี้เพียงคร่าว ๆ เนื่องจากในระบบกราฟิกส์ส่วนมากจะเตรียมคำสั่งในการระบายสีทึบให้แล้ว เราสามารถนำมาใช้ในโปรแกรมได้ทันที

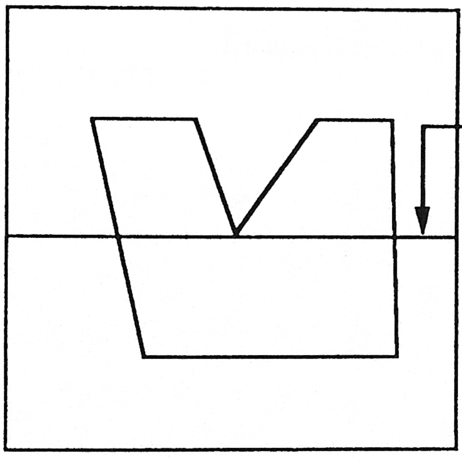


รูปที่ 4.5 การระบายสีทึบโดยวิธีเส้นสแกน

1. วิธีเส้นสแกน (scan-line algorithm) จะอาศัยจุดตัดของเส้นขอบเขตของรูปปิดกับเส้นสแกนเป็นตัวตัดสินว่าพิกเซลใดอยู่ภายในรูปปิด รูปที่ 4.5 แสดงวิธีการนี้ เส้นที่ลากผ่านรูปปิด

ก็คือเส้นสแกนซึ่งในรูปจะมีเพียงเส้นเดียว แต่ในการทำงานจริงจะต้องทำการสแกนหรือกวาดเส้นนี้ไปทั่วทั้งขอบเขตที่มีรูปปิดปรากฏอยู่เพื่อตรวจสอบว่าพิกเซลใดอยู่ภายในรูปปิด พิกเซลที่อยู่ภายในก็จะถูกทำให้สว่างขึ้นตามสีที่ถูกกำหนดไว้เมื่อตรวจสอบหมดแล้ว ผลที่ได้ก็คือการระบายสีรูปปิดนั่นเอง

จากรูปที่ 4.5 การตัดสินว่าพิกเซลใดอยู่ภายในหรือภายนอกทำได้ดังนี้ เส้นสแกนจะลากจากด้านซ้ายไปด้านขวาของจอภาพ การลากเส้นสแกนจากจุดทางซ้ายสุดก็คือ การพล็อตจุดหรือพิกเซลโดยใช้สีพื้นก่อน ถ้าไม่พบเส้นขอบเขตของรูปปิดเลย ซึ่งก็คือเส้นสแกนไม่ตัดกับเส้นขอบเขตก็ให้ลากเส้นสแกนในลำดับต่อไป (โดยทั่วไปลำดับจะเริ่มจากบนลงล่าง) แต่ถ้าหากเกิดจุดตัดขึ้น จุดที่พล็อตตรงจุดตัดและจุดที่จะพล็อตต่อมาจะใช้สีที่จะใช้ในการระบายรูปปิด ถ้าพล็อตไปแล้วพบจุดตัดอีกครั้ง ก็ให้เปลี่ยนสีการพล็อตเป็นสีพื้นและจะทำสลับกันไปเช่นนี้จนกว่าเส้นสแกนจะเกินขอบเขตที่รูปปิดอยู่



รูปที่ 4.6 ลักษณะรูปปิดซึ่งการใช้วิธีเส้นสแกนเกิดปัญหาในการระบายสี

วิธีการนี้มีปัญหาอยู่บ้างในกรณีที่ทำการระบายรูปปิดบางลักษณะ ดังแสดงในรูปที่ 4.6 เส้นสแกนจะตัดขอบเขตของรูปปิดเพียง 3 จุดเท่านั้น จะเห็นว่าเมื่อดำเนินวิธีการระบายสีทึบตามที่อธิบายข้างต้น จะมีบางส่วนที่ไม่ได้ระบาย กล่าวคือ หลังจากจุดตัดจุดแรกก็จะมีการระบายสี แต่เมื่อเกิดจุดตัดจุดที่ 2 ซึ่งความจริงควรจะระบายสีต่อ แต่ตามวิธีการข้างต้นจะไม่ระบายสี

2. วิธีเทสี (flood-fill algorithm) จะเริ่มจากการกำหนดจุดเริ่มระบายภายในขอบเขตของรูปปิดก่อน แล้วระบายจุดรอบ ๆ จุดเริ่มต้นขยายออกไปเรื่อย ๆ โดยตรวจสอบว่าจุดรอบ ๆ นั้นอยู่ภายในหรือภายนอกรูปปิด ถ้าอยู่ภายในก็ให้ระบายสี แต่ถ้าอยู่ภายนอกขอบเขตก็ไม่ต้องระบายสี

4.3.4 การระบายลวดลายและสีโดยใช้คำสั่งในเทอร์โบซี

เทอร์โบซีได้เตรียมคำสั่งเกี่ยวกับการระบายสีไว้หลายคำสั่ง ได้แก่ setfillstyle, setfillpattern และ floodfill นอกจากนี้ยังมีคำสั่งในการวาดรูปปิดพร้อมระบายลวดลายหรือสีให้เลย เช่น fillellipse สำหรับวาดรูปวงรีพร้อมระบายสีหรือลวดลายตามที่กำหนดไว้ก่อนแล้ว ส่วน fillpoly ใช้สำหรับวาดรูปเหลี่ยมพร้อมระบายสีหรือลวดลาย เป็นต้น

คำสั่ง setfillstyle ใช้สำหรับกำหนดรูปแบบลวดลายและสีที่จะนำไปประบายในรูปปิด โดยใช้ลวดลายที่ออกแบบไว้แล้ว ส่วนคำสั่ง setfillpattern ใช้สำหรับกำหนดรูปแบบลวดลายซึ่งผู้ใช้เป็นผู้ออกแบบเองกับสีที่ต้องการระบาย และ floodfill เป็นคำสั่งสำหรับการระบายสีโดยวิธีเทสี

สำหรับในหนังสือเล่มนี้จะไม่กล่าวถึงรายละเอียดของคำสั่งเหล่านี้ ผู้สนใจสามารถศึกษาได้จากคู่มือเทอร์โบซี ในที่นี้จะแสดงเพียงตัวอย่างการใช้งานคำสั่งเท่านั้น

คำสั่งที่กล่าวมาแล้วข้างต้นเป็นคำสั่งที่มีเฉพาะในเทอร์โบซีเท่านั้น ตัวแปลภาษาซีอื่น ๆ อาจจะมีคำสั่งเช่นนี้ด้วยแต่อาจจะใช้ชื่อคำสั่งต่างกัน ดังนั้นเพื่อให้คำสั่งที่ใช้ในตัวอย่างในหนังสือเล่มนี้ไม่ผูกพันกับเทอร์โบซี จึงกำหนดคำสั่งใหม่ดังนี้

Fill_Area	คำสั่งสำหรับระบายลวดลายหรือสีลงในรูปปิดโดยวิธีเทสี
Set_Fill_Pattern	คำสั่งสำหรับกำหนดรูปแบบลวดลายและสีซึ่งมีอยู่แล้วเพื่อที่จะใช้ในการระบายรูปปิด
Set_User_Fill_Pattern	คำสั่งสำหรับกำหนดรูปแบบลวดลายและสีที่ผู้ใช้ออกแบบเองเพื่อจะใช้ในการระบายรูปปิด

โปรแกรมที่ 4.5 การระบายลวดลายและสีลงในรูปปิด โดยใช้คำสั่งข้างต้น

```
/* A C program that demonstrates area attributes
*/

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
```

```
void Draw_Rectangle(int lt,int tp,int rt,int bt);
void Out_Text_XY(int x,int y,char text[]);
void Set_Fill_Pattern(int pattern,int color);
void Set_User_Fill_Pattern(char userpattern[],int color);
void Fill_Area(int x,int y,int color);


main()
{
    char pattern_1[8] = {0xF8,0xE0,0xF0,0xB8,0x9C,0x0E,0x07,0x02};
    char pattern_2[8] = {0x00,0x08,0x0C,0x0A,0x38,0x78,0x30,0x00};

    Initialize_Graphics_Mode();

    /* 1 solid fill */
    Draw_Rectangle(10,10,110,110);
    Set_Fill_Pattern(1,15);
    Fill_Area(60,60,15);
    Out_Text_XY(60,120,"1");

    /* 2 fill with --- */
    Draw_Rectangle(150,10,250,110);
    Set_Fill_Pattern(2,15);
    Fill_Area(200,60,15);
    Out_Text_XY(200,120,"2");

    /* 3 fill with ///,thick lines */
    Draw_Rectangle(300,10,400,110);
    Set_Fill_Pattern(4,15);
    Fill_Area(350,60,15);
    Out_Text_XY(350,120,"3");

    /* 4 fill with \\\ */
```

```
Draw_Rectangle(450,10,550,110);
Set_Fill_Pattern(6,15);
Fill_Area(500,60,15);
Out_Text_XY(500,120,"4");

/* 5 heavy crosshatch fill */
Draw_Rectangle(10,150,110,250);
Set_Fill_Pattern(8,15);
Fill_Area(60,200,15);
Out_Text_XY(60,260,"5");

/* 6 closely spaced dot fill */
Draw_Rectangle(150,150,250,250);
Set_Fill_Pattern(11,15);
Fill_Area(200,200,15);
Out_Text_XY(200,260,"6");

/* 7 fill with user defined pattern_1 */
Draw_Rectangle(300,150,400,250);
Set_User_Fill_Pattern(pattern_1,15);
Fill_Area(350,200,15);
Out_Text_XY(350,260,"7");

/* 8 fill with user defined pattern_2 */
Draw_Rectangle(450,150,550,250);
Set_User_Fill_Pattern(pattern_2,15);
Fill_Area(500,200,15);
Out_Text_XY(500,260,"8");

Out_Text_XY(150,300,"polygons with various attributes");

getchar();
```

```
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}
```

```
void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}

void Set_Fill_Pattern(int pattern,int color)
{
    setfillstyle(pattern,color);
}

void Set_User_Fill_Pattern(char userpattern[],int color)
{
    setfillpattern(userpattern,color);
}

void Fill_Area(int x,int y,int color)
{
    floodfill(x,y,color);
}
```

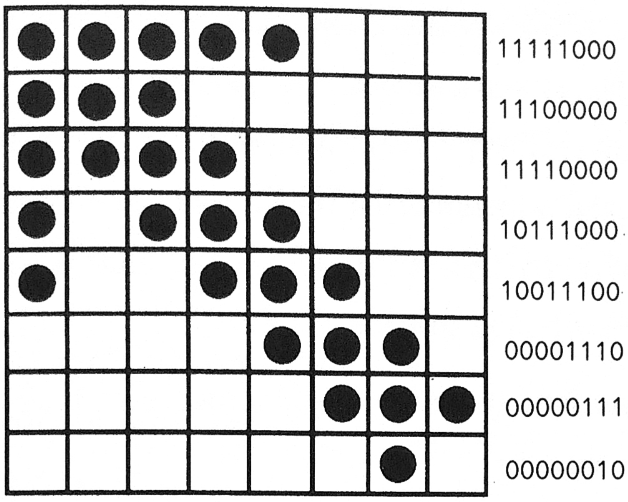
พารามิเตอร์ที่เราให้กับคำสั่ง Set__Fill__Pattern, Set__User__Fill__Pattern และ Fill__Area นั้น ได้มาจากพารามิเตอร์ที่ควรให้กับคำสั่ง setfillstyle, setfillpattern และ floodfill ของเทอร์โบซีนั่นเอง สำหรับข้อจำกัดและค่าของพารามิเตอร์ที่เหมาะสมกับคำสั่งนั้นเราสามารถศึกษาได้จากคู่มือเทอร์โบซี

กรณีที่ 7 และ 8 ในโปรแกรมที่ 4.5 นั้น เป็นการระบายลวดลายซึ่งผู้ใช้กำหนดเองซึ่งลวดลายนี้ก็คือ pattern__1 และ pattern__2 เป็นอาร์เรย์ของตัวอักษรและมีการกำหนดดังนี้

char pattern__1[8] = {0xF8,0xE0,0xF0,0xB8,0x9C,0x0E,0x07,0x02};

char pattern__2[8] = {0x00,0x08,0x0C,0x0A,0x38,0x78,0x30,0x00};

ค่าที่อยู่ภายในวงเล็บปีกกานั้นได้มาจากการออกแบบของเรานั่นเองในการออกแบบลวดลายเราจะใช้ตาราง ขนาด 8 x 8 ดังรูปที่ 4.7 ลวดลายที่เราต้องการออกแบบเป็นรูปลูกศรซึ่งแปลงเป็นค่าได้เป็นอาร์เรย์ชื่อ pattern__1



รูปที่ 4.7 ตารางออกแบบและลวดลาย pattern__1 ที่ออกแบบเสร็จแล้ว

ตารางที่ใช้ในการออกแบบอาจจะถือว่าเป็นส่วนเล็ก ๆ ของเฟรมบัฟเฟอร์โดยมีลักษณะเป็นสี่เหลี่ยมจัตุรัสขนาด 8 × 8 พิกเซลก็ได้ ช่องที่มีการระบายจุดคือพิกเซลที่ต้องการให้สว่าง ส่วนช่องที่ไม่มีการระบายจุดจะเป็นพิกเซลที่ไม่ต้องการให้สว่างนั่นเอง ช่องที่ระบายจุดก็จะรวมกันเป็นลวดลายแบบหนึ่งตามที่เราต้องการ เช่น กรณีนี้เป็นรูปลูกศร จากนั้นเราจะแปลงแบบลวดลายนี้ให้เป็นค่าเพื่อนำไปเป็นพารามิเตอร์ของคำสั่ง Set__User__Fill__Pattern หรือเป็นค่าที่จะกำหนดลงในอาร์เรย์ pattern__1 ข้างต้น

การแปลงแบบลวดลายให้เป็นค่าสำหรับกำหนดในอาร์เรย์จะทำดังนี้ ค่าในอาร์เรย์จะมีทั้งหมด 8 ค่า ซึ่งมาจาก 8 แถวของตารางนั่นเอง ค่าแรกคือค่าที่มาจากแถวบนสุดของตาราง ส่วนค่าสุดท้ายคือค่าที่มาจากแถวล่างสุดของตาราง พิจารณาค่าแรก 0×F8 ในอาร์เรย์ 0× เป็นตัวบอกว่าตัวเลขที่ตามมาเป็นเลขฐาน 16 นั่นคือ F8 เป็นเลขฐาน 16 และถ้าเขียนเลขจำนวนนี้เป็นเลขฐาน 2 จะเป็น 11111000 ซึ่งมี 8 หลัก ถ้าเอาเลขแต่ละหลักไปใส่ในช่องที่แถวแรกของตารางจะพบว่า ช่องที่มีการระบายจุดจะเป็น 1 ส่วนช่องที่ไม่มีการระบายจะเป็น 0 โดยวิธีการแปลงเช่นนี้เราก็จะได้แถวที่ 2 มีค่าเป็น E0 แถวที่ 3 มีค่าเป็น F0 และต่อ ๆ มาจนแถวสุดท้ายคือแถวที่ 8 จะมีค่าเป็น 02

สำหรับ pattern__2 ก็สามารถหาได้ในทำนองเดียวกันกับ pattern__1 ข้อจำกัดของการออกแบบลวดลายโดยวิธีนี้ก็คือ เราไม่สามารถออกแบบลวดลายเกินขนาดตาราง 8 × 8 ได้

4.4 แอตทริบิวต์ของตัวอักษร

แอตทริบิวต์ของตัวอักษรได้แก่ ฟอนต์ สี ขนาด การวางตำแหน่ง (orientation) ซึ่งได้แก่การวางในแนวนอน ในแนวดิ่ง ในแนวเฉียงหรือในแนวใด ๆ เป็นต้น สไตล์ของตัวอักษรซึ่งได้แก่

อักษรตัวธรรมดา ตัวหนา ตัวเอียง ตัวใหญ่ ตัวขีดเส้นใต้ และตัวยกกำลัง เป็นต้น นอกจากนี้ยังมีการกำหนดการวางแนวข้อความเวลาแสดงผล เช่น วางข้อความชิดขวา ชิดซ้าย หรือรักษาให้ข้อความอยู่ตรงกลางและอาจจะมีการวางข้อความชิดขอบบน ขอบล่างก็ได้ ระบบกราฟิกส์แต่ละระบบจะเตรียมคำสั่งในการกำหนดแอตทริบิวต์ให้แล้ว แต่ส่วนมากจะเตรียมไว้ให้เพียงบางอย่างเท่านั้น ตัวอย่างเช่น สำหรับเทอร์โบซีแอตทริบิวต์ที่ได้เตรียมให้แล้วได้แก่ ฟอนต์ ขนาด การวางตำแหน่งและการวางแนวข้อความเวลาแสดงผล เป็นต้น

คำสั่งเกี่ยวกับการกำหนดแอตทริบิวต์ของตัวอักษรในเทอร์โบซี ได้แก่ settextjustify, settextstyle และ setusercharsize

คำสั่ง settextjustify ใช้สำหรับกำหนดการวางข้อความให้ชิดขอบซ้าย ชิดขอบขวา หรือรักษาให้ข้อความอยู่ตรงกลางในแนวนอน และก็สามารถกำหนดการวางข้อความให้ชิดขอบบน ชิดขอบล่าง หรือรักษาให้ข้อความอยู่ตรงกลางในแนวดิ่ง

สำหรับคำสั่ง settextstyle จะใช้กำหนดฟอนต์ซึ่งมีให้เลือกทั้งแบบแรสเตอร์และแบบเวกเตอร์ (เทอร์โบซีจะเรียกตัวอักษรแบบแรสเตอร์ว่า ***บิตแมปฟอนต์ (bit-mapped font)*** ส่วนตัวอักษรแบบเวกเตอร์จะเรียกว่า ***สโตร๊กฟอนต์ (stroked font)***) นอกจากนี้จะกำหนดการวางตำแหน่งการแสดงข้อความว่าจะวางในแนวนอนหรือในแนวดิ่ง และยังกำหนดขนาดของตัวอักษรด้วยว่าต้องการขนาดเท่าไร

คำสั่ง setusercharsize จะใช้สำหรับการกำหนดขนาดของตัวอักษรซึ่งเป็นตัวอักษรแบบเวกเตอร์เท่านั้น สำหรับรายละเอียดของคำสั่งและค่าที่ใช้กำหนดในคำสั่งข้างต้นเหล่านี้ สามารถศึกษาได้จากคู่มือเทอร์โบซี

เนื่องจากคำสั่งที่กล่าวมาแล้วข้างต้นเป็นคำสั่งที่มีเฉพาะในเทอร์โบซี ดังนั้นเราจึงกำหนดคำสั่งใหม่ขึ้นมาใช้ในโปรแกรมตัวอย่างดังนี้

Set__Text__Attributes

ให้เป็นคำสั่งซึ่งรวมการทำงานของคำสั่ง settextjustify และ settextstyle

โปรแกรมที่ 4.6 แสดงข้อความการกำหนดแอตทริบิวต์โดยใช้คำสั่ง Set_Text_Attributes

```
/* A C program that demonstrates text attributes  */

#include <graphics.h>

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
int Get_Max_X(void);
int Get_Max_Y(void);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
void Set_Viewport(int lt,int tp,int rt,int bt);
void Clear_Viewport(void);
void Out_Text_XY(int x,int y,char text[]);
void Set_Text_Attributes(int font,int hor_ver,int size,
                         int horjust,int verjust);

main()
{
    Initialize_Graphics_Mode();

    Set_Text_Attributes(0,0,1,0,0);
    Out_Text_XY(Get_Max_X()/2,Get_Max_Y()/2,"  1st text string  ");

    Set_Text_Attributes(0,0,1,2,0);
    Out_Text_XY(Get_Max_X()/2,Get_Max_Y()/2,"  2st text string  ");

    Set_Text_Attributes(3,1,0,0,0);
    Out_Text_XY(Get_Max_X()/2,Get_Max_Y()/2,"  3nd text string  ");

    Set_Text_Attributes(3,1,0,0,2);
    Out_Text_XY(Get_Max_X()/2,Get_Max_Y()/2,"  4st text string  ");
```

```
    Set_Text_Attributes(0,0,1,0,0);
    Out_Text_XY(200,Get_Max_Y()-20,"text with various attributes");
    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Out_Text_XY(int x,int y,char text[])
```

```
{
    outtextxy(x,y,text);
}

int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}

void Set_Viewport(int lt,int tp,int rt,int bt)
{
    setviewport(lt,tp,rt,bt,1);
}

void Clear_Viewport(void)
{
    clearviewport();
}

void Set_Text_Attributes(int font,int hor_ver,int size,
                         int horjust,int verjust)
{
```

```
    settextstyle(font,hor_ver,size);
    settextjustify(horjust,verjust);
}
```

4.5 สรุป

ภาพกราฟิกส์พื้นฐาน เช่น จุด เส้น รูปปิด และตัวอักษรหรือข้อความนั้น เราสามารถเพิ่มคุณสมบัติอื่น ๆ ให้กับคำสั่งสร้างภาพกราฟิกส์พื้นฐานเหล่านี้ได้ คุณสมบัติเพิ่มเติมเหล่านี้จะเรียกว่าแอตทริบิวต์ของคำสั่ง แอตทริบิวต์ของคำสั่งสร้างจุดก็คือ *สี* แอตทริบิวต์ของคำสั่งสร้างเส้นได้แก่ *แบบของเส้น สี* และ *ความหนาของเส้น* แอตทริบิวต์ของคำสั่งสร้างรูปปิด ได้แก่ *การระบายสีหรือลวดลายภายในรูปปิด* แอตทริบิวต์ของคำสั่งสร้างตัวอักษรหรือข้อความ ได้แก่ *ฟอนต์ สี ขนาดการวางตำแหน่งข้อความ* และ *การวางแนวข้อความให้ชิดขอบเขต*

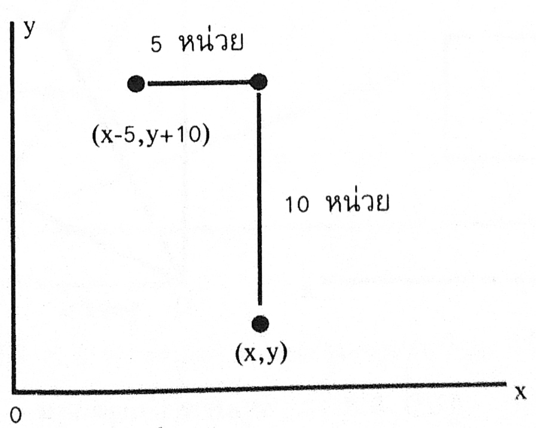
5 การแปลงภาพเรขาคณิต 2 มิติ

5.1 บทนำ

การแปลงภาพเรขาคณิต (geometric transformations) หมายถึง ภาพที่วาดขึ้นมานั้นสามารถที่จะนำมาเปลี่ยนแปลง แก้ไข ตำแหน่งหรือขนาดของมันได้ ซึ่งการแปลงที่จะกล่าวถึงในบทนี้ได้แก่ การย้ายภาพ (translation), การหมุนภาพ (rotation), การย่อขยายภาพ(scaling) และการบิดภาพ (shearing)

5.2 การย้ายภาพ

จุดใด ๆ ก็ตามในระบบพิกัดโลกจะถูกเคลื่อนย้ายไปที่ตำแหน่งอื่น ๆ ได้โดยการเปลี่ยนค่าพิกัด (x,y) ของมัน ตัวอย่างเช่น ในรูปที่ 5.1 เราต้องการย้ายจุด (x,y) ไปยังจุดซึ่งอยู่ข้างบน 10 หน่วยและไปทางซ้ายอีก 5 หน่วย พิกัดของจุดใหม่ที่ได้ก็คือ (x-5,y+10) ซึ่งก็คือการย้ายจุด (x,y) นั่นเอง



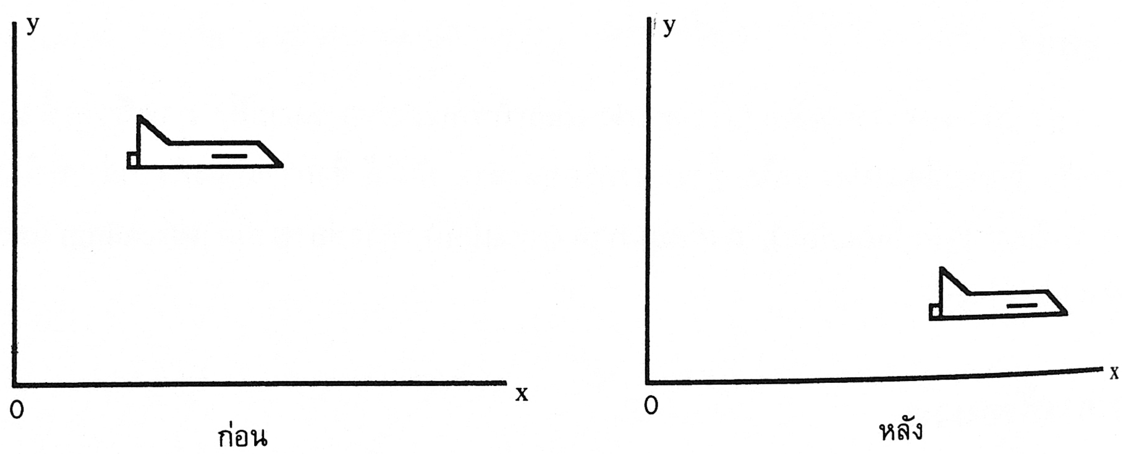
รูปที่ 5.1 การย้ายจุด (x,y)

โดยทั่วไปการย้ายจุดจากจุด (x,y) ไปยังจุด (x',y') โดยย้ายในแนวนอน H หน่วย และย้ายในแนวตั้ง V หน่วย สามารถเขียนแทนด้วยสูตรทางคณิตศาสตร์ดังนี้

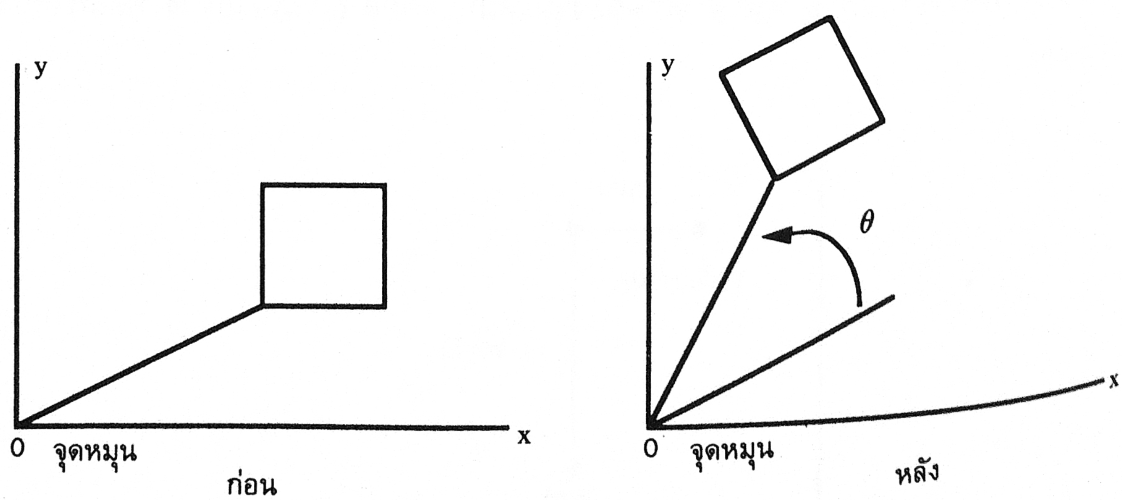
$$x' = x + H$$

$$y' = y + V$$

ในทางปฏิบัติจริง ๆ นั้น เราจะต้องย้ายทุก ๆ จุดที่ใช้สำหรับนิยามภาพจึงจะสามารถย้ายภาพทั้งหมดไปยังจุดที่ต้องการได้ รูปที่ 5.2 แสดงการย้ายภาพเครื่องบินจากตำแหน่งหนึ่งไปยังอีกตำแหน่งหนึ่ง จุดทุกจุดที่ใช้สำหรับนิยามภาพจะถูกย้ายไปยังตำแหน่งที่ต้องการก่อนแล้วจึงทำการวาดภาพนั้นใหม่ที่ตำแหน่งนั้น



รูปที่ 5.2 การย้ายภาพ



รูปที่ 5.3 การหมุนภาพรอบจุดหมุน

5.3 การหมุนภาพ

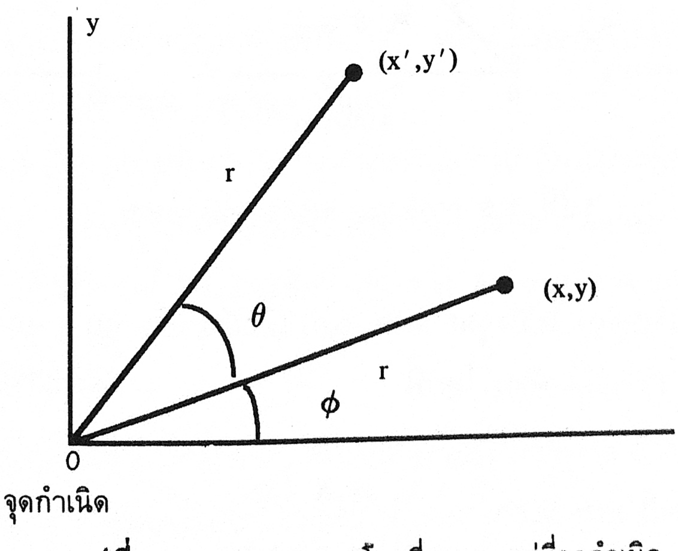
การหมุนภาพเป็นการแปลงอีกแบบหนึ่ง สำหรับการหมุนภาพนี้จะต้องกำหนดว่าจุดใดเป็นจุดหมุนเสมอ รูปที่ 5.3 แสดงการหมุนภาพสี่เหลี่ยมรอบจุดหมุน (pivot point) ซึ่งอยู่ที่จุดกำเนิดหรือจุด (0,0) นั่นเอง หลังจากที่ภาพถูกหมุนไปแล้ว ระยะห่างระหว่างจุดหมุนกับภาพจะยังคงมีค่าเท่าเดิม รูปร่าง ลักษณะของภาพก็ยังคงเดิม แต่ภาพจะมีการจัดวางที่ต่างไปจากเดิมอันเนื่องมาจากการหมุนนั่นเอง การหมุนภาพนี้อาจจะหมุนทีละหลาย ๆ ภาพก็ได้ จะหมุนแบบทวนเข็มนาฬิกาหรือตามเข็มนาฬิกาก็ได้ และจุดหมุนที่ใช้ก็อาจจะอยู่ภายในภาพหรืออยู่ภายนอกภาพก็ได้

การอ้างถึงจุดพิกัด (x,y) นั้น นอกจากจะใช้ระบบพิกัดฉากแล้ว (คือกำหนดตำแหน่งจุดโดยบอกระยะทางในแนวนอนและในแนวดิ่ง) เราอาจจะใช้ระบบพิกัดโพลาร์ก็ได้ (ระบบพิกัดโพลาร์คือการบอกตำแหน่งจุดโดยใช้เวกเตอร์) สองระบบนี้มีความสัมพันธ์กันดังนี้

$$x = r\cos(\phi)$$
$$y = r\sin(\phi)$$

ถ้า (x,y) ถูกหมุนไปจากจุดเดิมเป็นมุม θ ในทิศทางทวนเข็มนาฬิกาจะได้จุดใหม่คือ (x′,y′) ดังรูปที่ 5.4 เราจะได้ว่า

$$x' = r\cos(\phi+\theta)$$
$$y' = r\sin(\phi+\theta)$$



รูปที่ 5.4 การหมุนภาพโดยที่จุดหมุนอยู่ที่จุดกำเนิด

ซึ่งอาจเขียนใหม่ได้ดังนี้

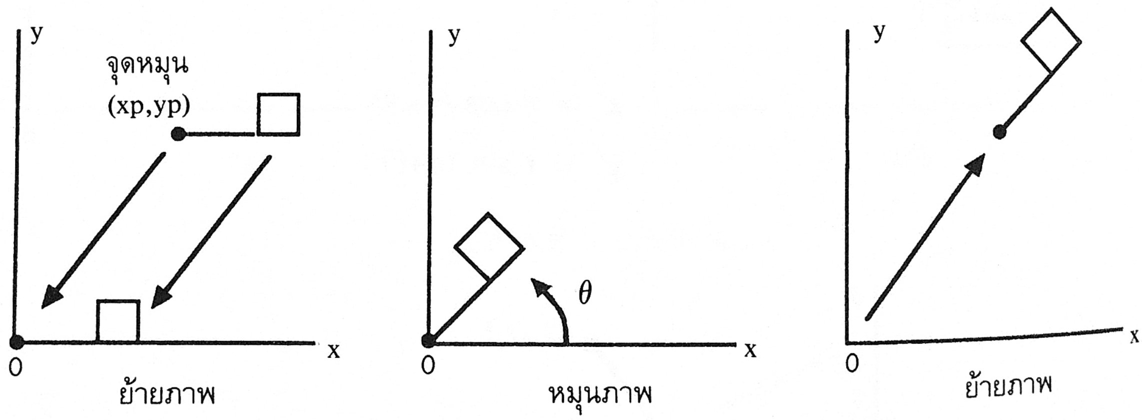
$$x' = r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$$
$$y' = r\sin(\phi)\cos(\theta) + r\cos(\phi)\sin(\theta)$$

และแทนค่า $r\cos(\phi)$ ด้วย x, $r\sin(\phi)$ ด้วย y จะได้

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = y\cos(\theta) + x\sin(\theta)$$

สมการข้างต้นก็คือสมการที่ใช้แปลงค่าพิกัด จากจุด (x,y) ไปเป็นจุดใหม่ (x′,y′) โดยการหมุนรอบจุดกำเนิดไปเป็นมุม θ ทิศทางทวนเข็มนาฬิกา การหมุนภาพทำได้โดยการแปลงค่าพิกัดทุกจุดที่ใช้นิยามภาพนั้นไปเป็นพิกัดใหม่โดยใช้สมการข้างต้นแล้วค่อยวาดภาพเดิมที่จุดพิกัดใหม่นั้น

ในทางปฏิบัติเราอาจจะต้องหมุนภาพรอบจุดใด ๆ ก็ได้ซึ่งไม่ใช่จุดกำเนิด สำหรับในกรณีนี้จะต้องใช้ 3 ขั้นตอนดังนี้ (รูปที่ 5.5)



รูปที่ 5.5 การหมุนภาพรอบจุดหมุนใด ๆ

1. ย้ายจุดหมุน (xp,yp) ไปยังจุดกำเนิด (0,0) เมื่อย้ายแล้ว ทุก ๆ จุด (x,y) ที่ใช้นิยามภาพก็จะถูกย้ายไปยังจุดใหม่ (x′,y′) ด้วย โดยที่

$$x' = x - xp$$
$$y' = y - yp$$

ในตอนนี้จุดหมุนก็จะถูกย้ายไปยังจุดกำเนิด (0,0)

2. จัดการหมุนภาพรอบจุดกำเนิด นั่นคือจุด (x′,y′) ถูกย้ายไปเป็นมุม θ ได้จุดใหม่เป็น (x″,y″) โดยที่

$$x'' = x' \cos(\theta) - y' \sin(\theta)$$
$$y'' = y' \cos(\theta) + x' \sin(\theta)$$

แทนค่า x′ และ y′ จะได้

$$x'' = (x - xp) \cos(\theta) - (y - yp) \sin(\theta)$$
$$y'' = (y - yp) \cos(\theta) + (x - xp) \sin(\theta)$$

3. ย้ายจุดหมุนจากจุดกำเนิด (0,0) กลับไปยังจุดเดิม (xp,yp) ดังนั้นจุด (x″,y″) ก็จะถูกย้ายไปยังจุด (x*,y*) โดยที่

$$x^* = x'' + xp$$
$$y^* = y'' + yp$$

แทนค่า x″ และ y″ จะได้ว่า

$$x^* = (x - xp) \cos(\theta) - (y - yp) \sin(\theta) + xp$$
$$y^* = (y - yp) \cos(\theta) + (x - xp) \sin(\theta) + yp$$

สมการนี้คือสมการสำหรับหาค่าพิกัดใหม่ เมื่อหมุนภาพไปเป็นมุม θ ในทิศทางทวนเข็มนาฬิกา โดยหมุนรอบจุดหมุน (xp,yp) นั่นเอง

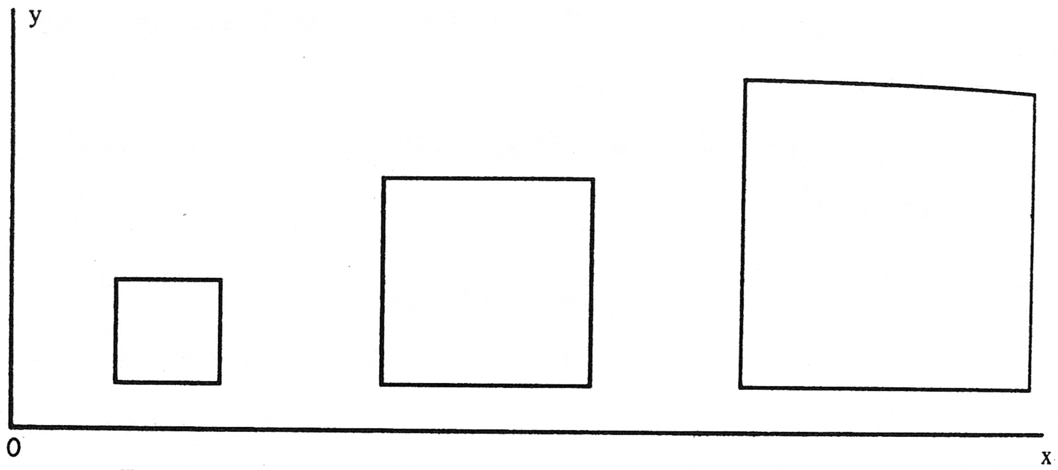
เรามักจะนิยมทำการหมุนภาพรอบจุดโดยหมุนทวนเข็มนาฬิกาเสมอ เนื่องจากว่าในทางคณิตศาสตร์จะกำหนดว่า ทิศทางการหมุนที่เป็นบวกจะเป็นทิศทวนเข็มนาฬิกา ถ้าเราต้องการจะหมุนตามเข็มนาฬิกาก็สามารถทำได้โดยใช้ค่า $-\theta$ แทน θ ในสมการ ผลที่ได้ก็จะเหมือนกัน

5.4 การย่อขยายภาพ

เราสามารถเปลี่ยนขนาดของภาพได้โดยการเปลี่ยนขนาดหน้าต่างแสดงภาพหรือเปลี่ยนขนาดของช่องแสดงภาพ เทคนิคนี้ไม่สามารถใช้ได้ในบางกรณี เช่น ถ้าเราต้องการเปลี่ยนขนาดของภาพภาพหนึ่งในหน้าต่างเท่านั้น ถ้าเราขยายหรือย่อขนาดของหน้าต่าง ภาพทั้งหมดในหน้าต่างก็จะขยายหรือย่อไปด้วย ในบทนี้เราจะกล่าวถึงวิธีการเปลี่ยนขนาดของภาพเฉพาะภาพใดภาพหนึ่ง

ภาพวัตถุภาพหนึ่งสามารถเปลี่ยนขนาดได้โดยการเปลี่ยนระยะห่างระหว่างจุด (รูปที่ 5.6) โดยทั่วไป เราสามารถเปลี่ยนขนาดของภาพวัตถุได้โดยการคูณระยะห่างระหว่างจุดด้วยค่าซึ่งทำระยะห่างมากขึ้นหรือทำให้ระยะห่างลดลง ค่านี้เราเรียกว่า *สเกลลิงแฟกเตอร์ (scaling factor)* ถ้าค่าสเกลลิงแฟกเตอร์มากกว่า 1 ก็จะได้ภาพขยาย ถ้าค่านี้น้อยกว่า 1 ก็จะได้เป็นภาพย่อ ถ้าเท่ากับ 1 ก็หมายถึงไม่มีผลต่อภาพวัตถุ

เมื่อใดก็ตามที่มีการย่อหรือขยายภาพ จะต้องมีจุด ๆ หนึ่งซึ่งจะเรียกว่าเป็นจุดประจำที่ (fixed point) ของการย่อขยายภาพซึ่งใช้สำหรับเป็นจุดอ้างอิง



รูปที่ 5.6 การย่อขยายภาพ

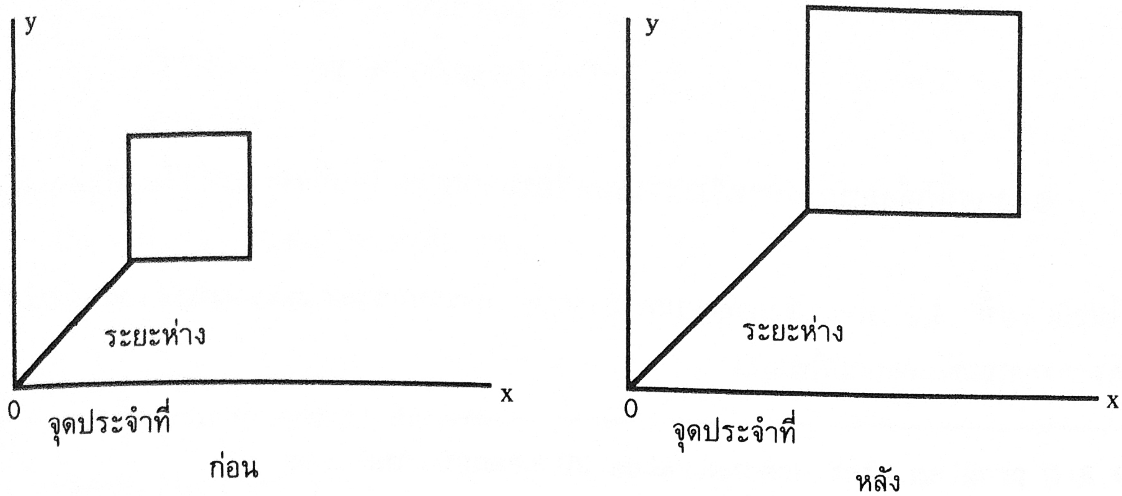
ถ้าให้จุดกำเนิด (0,0) เป็นจุดประจำที่ จุด (x,y) ใด ๆ ของภาพ ก็จะสามารถย่อหรือขยายได้โดยการคูณด้วยแฟกเตอร์ Sx สำหรับทิศทางในแกน x และแฟกเตอร์ Sy สำหรับทิศทางในแกน y ก็จะได้จุดใหม่ (x′,y′) ดังนี้

$$x' = x \times Sx$$
$$y' = y \times Sy$$

Sx คือสเกลลิงแฟกเตอร์ในแนวนอน

Sy คือสเกลลิงแฟกเตอร์ในแนวดิ่ง

ถ้า Sx ไม่เท่ากับ Sy ผลก็คือภาพที่ได้จากการย่อหรือขยายจะเกิดการผิดเพี้ยนไปจากภาพเดิม ถ้าสเกลลิงแฟกเตอร์มากกว่า 1 ภาพที่ถูกขยายแล้ว จะถูกเคลื่อนย้ายห่างออกไปจากจุดประจำที่ ถ้าสเกลลิงแฟกเตอร์น้อยกว่า 1 ภาพที่ถูกย่อแล้ว จะถูกเคลื่อนย้ายเข้ามาใกล้กับจุดประจำที่มากขึ้น รูปที่ 5.7 แสดงปรากฏการณ์นี้



รูปที่ 5.7 การย่อขยายภาพทำให้ระยะห่างระหว่างภาพกับจุดประจำที่เปลี่ยนไป

จุดประจำที่สำหรับการย่อขยายภาพอาจจะไม่ใช่จุดกำเนิดก็ได้ ถ้าเป็นจุดใด ๆ เราจะต้องใช้ 3 ขั้นตอนดังนี้

1. ย้ายจุดประจำที่ (xp,yp) ไปยังจุดกำเนิด จุดอื่น ๆ ของภาพ (x,y) ก็จะถูกย้ายไปที่จุดใหม่ (x′,y′) และ

$$x' = x - xp$$
$$y' = y - yp$$

2. จัดการย่อหรือขยายภาพโดยที่จุดประจำที่อยู่ที่จุดกำเนิด ก็จะได้จุดที่ใช้นิยามภาพจุดใหม่เป็น (x″,y″)

$$x'' = x' \times Sx$$
$$y'' = y' \times Sy$$

3. ย้ายจุดประจำที่ จากจุดกำเนิดไปยังจุดประจำที่จุดเดิม (xp,yp)

$$x^* = x'' + xp$$
$$y^* = y'' + yp$$

แทนค่า x′,y′,x″ และ y″ จะได้ว่า

$$x^* = (x\text{-}xp)Sx + xp$$

$$y^* = (y\text{-}yp)Sy + yp$$

สมการนี้ก็คือสมการสำหรับการย่อหรือขยายภาพ โดยที่จุดประจำที่อยู่ที่จุด (xp,yp)

โปรแกรมที่ 5.1 การแปลงภาพแบบการย้ายภาพ การหมุนภาพ และการย่อขยายภาพโดยใช้สมการการแปลงแต่ละแบบโดยตรง

```
/* A C program that demonstrates 2D transformations
 * translation, rotation and scaling using their formulae
*/

#include <graphics.h>
#include <math.h>

#define NUM_OF_POINTS 50
#define PI 3.14159

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Out_Text_XY(int x,int y,char text[]);
void Clear_Viewport(void);

main()
{
    struct coord {  /* structure of points which define image */
         int x;
         int y;
    };
    struct coord imagedef[NUM_OF_POINTS] = { /* image definition */
         100,100,
```

```
    100,0,
    0,100,
    100,100
};
int i,h,v,new_x,new_y,xp,yp;
float degree,Sx,Sy;

Initialize_Graphics_Mode();
Clear_Viewport();

/* draw the image */
Move_To(0,0);
for(i=0; i <= 4; i++) {
      Draw_Line_To(imagedef[i].x,imagedef[i].y);
}
Out_Text_XY(5,110,"original image");
getchar();

Clear_Viewport();
/* translate the image to new position */
/* 200 units in horizontal and 150 units in vertical */
h = 200; v = 150;
/* draw the image in new position */
Move_To(0+h,0+v);
for(i=0; i <= 4; i++) {
      Draw_Line_To(imagedef[i].x+h,imagedef[i].y+v);
}
Out_Text_XY(5+h,110+v,"translated image");
getchar();

Clear_Viewport();
/* rotate the image around the point
```

```
   (200,150) 90 degrees counterclockwise of screen coordinate */
degree = PI/2; xp = 200; yp =150;
Move_To(xp,yp);
/* draw the image in new position after rotate */
for(i=0; i <= 4; i++) {
        new_x = (imagedef[i].x+h-xp)*cos(degree)
              -(imagedef[i].y+v-yp)*sin(degree)+xp;
        new_y = (imagedef[i].y+v-yp)*cos(degree)
              +(imagedef[i].x+h-xp)*sin(degree)+yp;
Draw_Line_To(new_x,new_y);
}
Out_Text_XY(xp-100,yp-10,"rotated image");
getchar();

Clear_Viewport();
/* scale the image */
/* fixed point is (300,50)
 * and scaling factor Sx = Sy = 2 to enlarge the image
 * Sx = Sy = 0.5 to make the image smaller */
/* enlarging image */
xp = 300; yp = 50; Sx = 2; Sy = 2;
Move_To(xp,yp);  /* fixed point */
for(i=0; i <= 4; i++) {
      new_x = imagedef[i].x*Sx + xp;
      new_y = imagedef[i].y*Sy + yp;
      Draw_Line_To(new_x,new_y);
}
Out_Text_XY(xp,yp,"enlarged image");
getchar();

/* reducing image */
xp = 50; yp = 300; Sx = 0.5; Sy = 0.5;
```

```
    Move_To(xp,yp);  /* fixed point */
    for(i=0; i <= 4; i++) {
          new_x = imagedef[i].x*Sx + xp;
          new_y = imagedef[i].y*Sy + yp;
          Draw_Line_To(new_x,new_y);
    }
    Out_Text_XY(xp,yp,"reduced image");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
          printf("Graphics error : %s\n", grapherrormsg(errorcode));
          printf("Press any key to halt : ");
          getch();
          exit(1); /* return with error code */
    }
}
```

```
void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}
void Move_To(int x,int y)
{
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Clear_Viewport(void)
{
    clearviewport();
}
```

5.5 การบิดภาพ

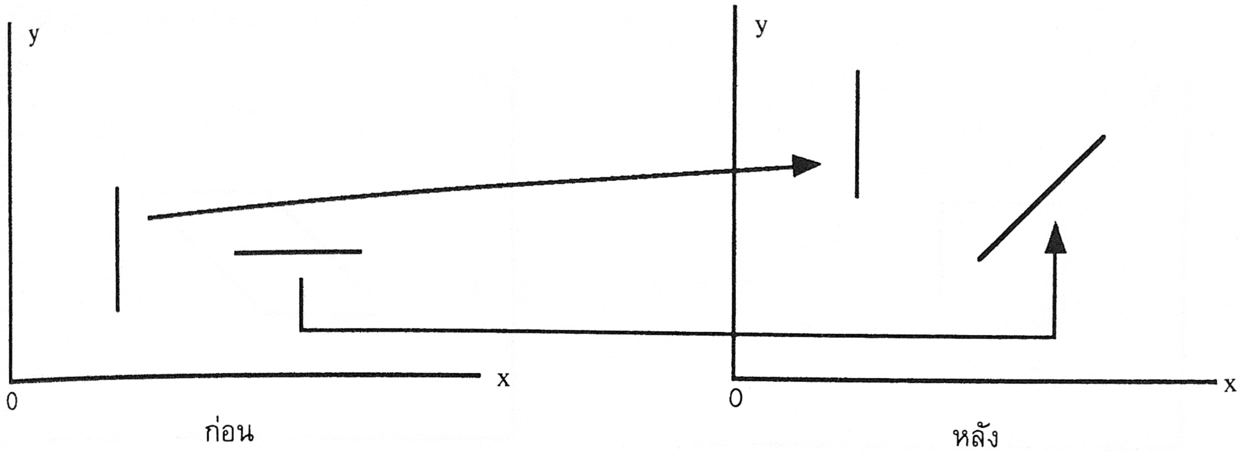
การบิดภาพจะทำให้บางส่วนของภาพหรือภาพทั้งหมดเกิดการบิดเบือนขึ้น ในที่นี้เราจะพิจารณาเพียง 2 แบบคือ การบิดภาพทางแกน x และการบิดภาพทางแกน y

การบิดภาพทางแกน y จะทำให้เกิดการย้ายจุด (x,y) ไปยังจุด (x′,y′) โดยที่

$$x' = x$$

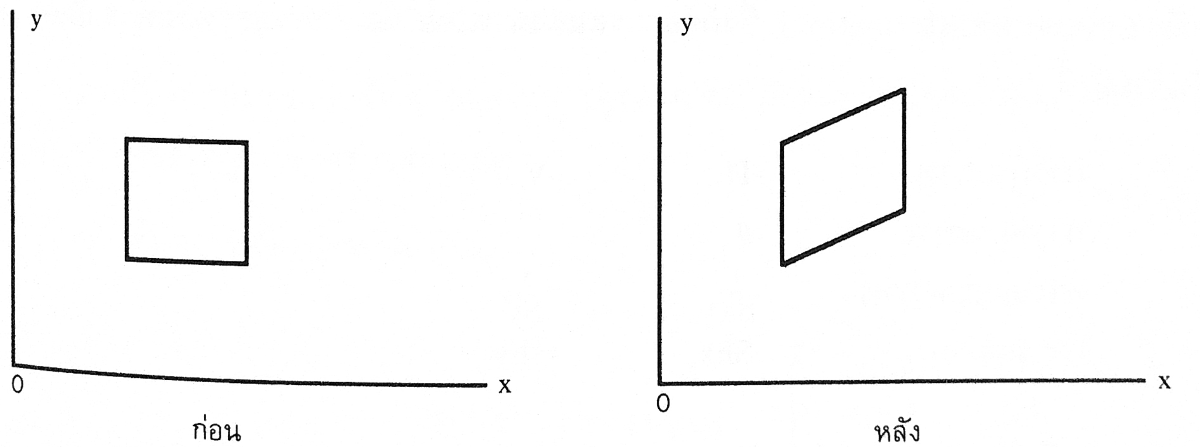
$$y' = Shy \times x + y,\ Shy \text{ ไม่เท่ากับ } 0$$

การบิดภาพทางแกน y จะทำให้จุดต่าง ๆ ในแกน y เลื่อนขึ้นหรือเลื่อนลงขึ้นอยู่กับเครื่องหมายของแฟกเตอร์ Shy (รูปที่ 5.8) เส้นตรงในแนวนอนจะถูกเปลี่ยนให้เป็นเส้นตรงในแนวเฉียงด้วยความลาดชันเท่ากับ Shy



รูปที่ 5.8 การบิดภาพทางแกน y ทำให้เส้นในแนวนอนเปลี่ยนไปเป็นเส้นในแนวเฉียง

รูปที่ 5.9 แสดงการบิดภาพสี่เหลี่ยมทางแกน y ซึ่งทำให้ภาพเดิมซึ่งเป็นภาพสี่เหลี่ยมผืนผ้าเปลี่ยนแปลงไปเป็นภาพสี่เหลี่ยมด้านขนาน



รูปที่ 5.9 การบิดภาพสี่เหลี่ยมทางแกน y

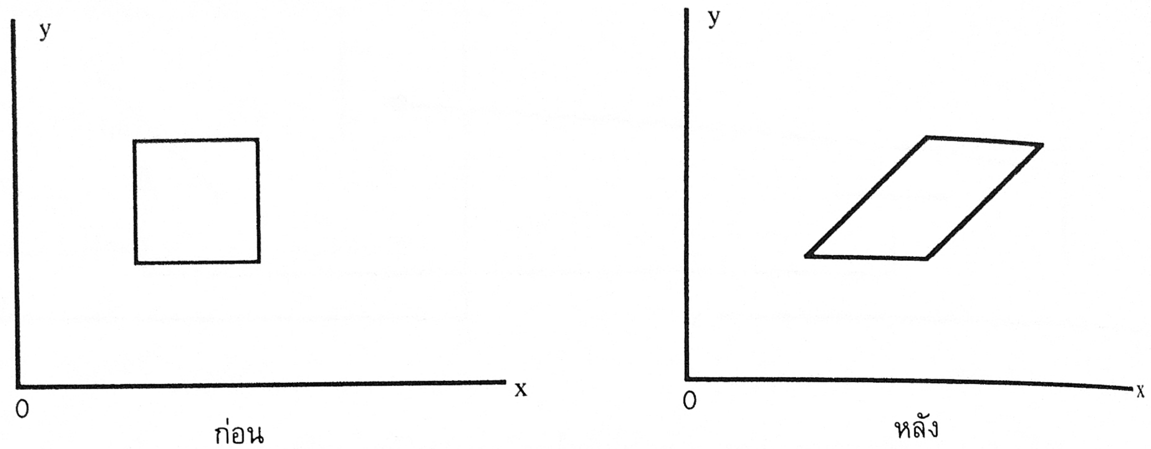
สำหรับการบิดภาพทางแกน x จะให้ผลตรงข้ามกับการบิดภาพทางแกน y กล่าวคือ จุด (x,y) ของภาพจะถูกแปลงเป็นจุด (x′,y′) โดยที่

$$x' = x + Shx \times y,\ Shx \text{ ไม่เท่ากับ } 0$$

$$y' = y$$

ในกรณีนี้เส้นในแนวนอนก็จะถูกย้ายไปทางซ้ายหรือขวา ขึ้นอยู่กับเครื่องหมายของแฟกเตอร์ Shx ส่วนเส้นตรงในแนวดิ่งก็จะถูกบิดไปเป็นเส้นตรงในแนวเฉียงด้วยความลาดชัน Shx

รูปที่ 5.10 เป็นการบิดภาพสี่เหลี่ยมทางแกน x ภาพจะเปลี่ยนจากสี่เหลี่ยมผืนผ้าให้กลายเป็นสี่เหลี่ยมด้านขนาน



รูปที่ 5.10 การบิดภาพสี่เหลี่ยมทางแกน x

5.6 การแปลงภาพกลับ

เมื่อเราแปลงภาพไปแล้ว ถ้าเราต้องการให้ได้ภาพเดิมกลับมาเราทำได้โดยการแปลงภาพกลับ (inverse transformation) ซึ่งก็คือการแปลงภาพโดยใช้ค่าแฟกเตอร์หรือพารามิเตอร์ตรงกันข้ามดังนี้

การย้ายภาพ	:	-H,	-V
การหมุนภาพ	:	$-\theta$	
การย่อขยายภาพ	:	$\frac{1}{Sx}$,	$\frac{1}{Sy}$
การบิดภาพ	:	-Shx,	-Shy

5.7 การใช้แมทริกซ์สำหรับการแปลงภาพ

สมการสำหรับการแปลงแต่ละแบบที่กล่าวมาแล้วสามารถเขียนอยู่ในรูปของแมทริกซ์ได้ การที่เราใช้รูปของแมทริกซ์สำหรับการคำนวณก็เนื่องมาจากเป็นรูปแบบที่เหมาะกับการคำนวณโดยใช้คอมพิวเตอร์และจะทำการคำนวณได้ง่ายขึ้นเมื่อนำเอาการแปลงหลาย ๆ แบบมาผสมกัน

เราจะคำนวณหาจุดใหม่ที่ได้จากการแปลงแบบต่าง ๆ โดยการคูณเวกเตอร์ขนาด 1×3 กับแมทริกซ์ที่เหมาะสมขนาด 3×3 เวกเตอร์ขนาด 1×3 ใช้แทนค่าพิกัดของจุด (x,y) ส่วนแมทริกซ์ 3×3 จะแทนแฟกเตอร์หรือพารามิเตอร์ของการแปลง

เนื่องจากเราต้องการใช้การคำนวณทางแมทริกซ์มาใช้ในการคำนวณการแปลงภาพ เราจึงต้องแปลงพิกัดปกติของเราให้เป็น *พิกัดโฮโมจีเนียส (homogeneous coordinates)* ก่อน นั่นคือถ้าเป็นพิกัด 2 มิติก็จะต้องเขียนในรูปเวกเตอร์ 1 × 3 เช่น พิกัด (x,y) เมื่อเขียนในรูปของเวกเตอร์จะเป็น [x y 1] สำหรับจุดใหม่ที่แปลงแล้วก็คือ (x′,y′) เขียนในรูปของเวกเตอร์จะเป็น [x′ y′ 1] และสำหรับแมทริกซ์ของการแปลงแต่ละแบบจะเป็นดังนี้

การย้ายภาพ :

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{bmatrix}$$

ผลของการคูณเวกเตอร์ [x y 1] กับแมทริกซ์ 3 × 3 จะเป็นดังนี้

$$x' = x + H$$
$$y' = y + V$$
$$1 = 1$$

ซึ่งก็คือสมการสำหรับการย้ายภาพนั่นเอง

สำหรับการแปลงแบบอื่น ๆ ผลของการคูณเวกเตอร์กับแมทริกซ์ก็จะได้สมการของการแปลงนั้น ๆ ซึ่งขอให้ผู้ศึกษาตรวจสอบเอง

การหมุนภาพรอบจุดหมุน (0,0) :

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

การหมุนภาพรอบจุดหมุน (Xr,Yr) :

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ (1-\cos(\theta))Xr & (1-\cos(\theta))Yr & 1 \\ +Yr\sin(\theta) & -Xr\sin(\theta) & \end{bmatrix}$$

การย่อขยายภาพจุดประจำที่อยู่ที่ (0,0) :

$$[x' \; y' \; 1] = [x \; y \; 1] \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

การย่อขยายภาพจุดประจำที่อยู่ที่ (Xf,Yf) :

$$[x' \; y' \; 1] = [x \; y \; 1] \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ (1-Sx)Xf & (1-Sy)Yf & 1 \end{bmatrix}$$

การบิดภาพทางแกน y :

$$[x' \; y' \; 1] = [x \; y \; 1] \begin{bmatrix} 1 & Shy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

การบิดภาพทางแกน x :

$$[x' \; y' \; 1] = [x \; y \; 1] \begin{bmatrix} 1 & 0 & 0 \\ Shx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

สำหรับแมทริกซ์ที่ใช้ในการแปลงภาพกลับก็ใช้แมทริกซ์ข้างต้นนั่นเอง เพียงแต่ค่าต่าง ๆ ซึ่งเป็นสมาชิกของแมทริกซ์จะเปลี่ยนไปใช้ค่าตรงข้ามซึ่งได้กล่าวไว้แล้วในตอนที่ 5.6

5.8 การแปลงภาพหลาย ๆ แบบผสมกัน

การแปลงภาพแบบต่าง ๆ ที่กล่าวมาแล้วข้างต้นสามารถนำมาผสมกันกลายเป็นการแปลงภาพแบบใหม่ได้ เช่น ทำการหมุนภาพพร้อม ๆ กับการย้ายภาพ สมมติว่าเราทำให้เกิดเป็นภาพเคลื่อนไหวได้ เราจะได้ภาพนั้นหมุนไปพร้อม ๆ กับเคลื่อนที่ไปด้วยนั่นเอง

การคำนวณหาจุดใหม่ที่ได้จากการแปลงภาพแบบนี้ก็คือ จะต้องใช้สมการสำหรับการหมุนภาพและสมการสำหรับการย้ายภาพ ซึ่งถ้าใช้แมทริกซ์เราก็จะสามารถเขียนสมการซึ่งรวมสมการการแปลงภาพทั้งสองแบบนี้ได้ง่ายขึ้น ดังนี้

$$[x' \; y' \; 1] = [x \; y \; 1] \times Rot \times Tr$$

Rot คือแมทริกซ์ของการหมุนภาพซึ่งกล่าวไว้แล้วข้างต้น

Tr คือแมทริกซ์ของการย้ายภาพซึ่งกล่าวไว้แล้วข้างต้น

การคำนวณหาค่าจุดใหม่ (x′,y′) ทำได้ 2 วิธี วิธีแรกทำการคูณ [x y 1] กับ Rot ก่อน ซึ่งจะต้องใช้การคูณ 9 ครั้ง กับการบวกอีก 6 ครั้ง ดังนี้

$$[x'' \; y'' \; 1] = [x \; y \; 1] \times Rot$$

[x″ y″ 1] เป็นเวกเตอร์ผลของการคูณ

จากนั้นก็นำผลที่ได้ไปคูณกับ Tr จึงจะได้ผลลัพธ์สุดท้ายเป็นจุดใหม่ ซึ่งในขั้นตอนนี้ก็ต้องใช้การคูณ 9 ครั้งกับการบวก 6 ครั้งเช่นเดียวกัน ดังนี้

$$[x' \; y' \; 1] = [x'' \; y'' \; 1] \times Tr$$

การคำนวณทั้งหมดของวิธีนี้จะใช้การคูณรวม 18 ครั้งกับการบวกรวม 12 ครั้ง จึงจะได้จุดใหม่ที่แปลงแล้ว 1 จุด ถ้าภาพที่ทำการแปลงประกอบด้วยจุดหลาย ๆ จุด การคำนวณวิธีนี้ก็จะกินเวลามาก

วิธีที่ 2 จะสามารถลดจำนวนการคำนวณลงได้ โดยทำการคูณ Rot กับ Tr เสียก่อน ซึ่งต้องใช้การคูณ 27 ครั้งกับการบวก 18 ครั้ง ซึ่งดูเหมือนจะมากกว่าการคำนวณในแบบแรกถ้าเป็นการแปลงจุดเพียงจุดเดียว การคูณ Rot กับ Tr นี้จะทำเพียงครั้งเดียวเท่านั้น จากนั้นจะนำผลของการคูณนี้ไปคูณกับจุด [x y 1] ดังนี้

$$[x' \; y' \; 1] = [x \; y \; 1] \times (Rot \times Tr)$$

ขั้นตอนนี้จะใช้การคูณ 9 ครั้งกับการบวก 6 ครั้ง ซึ่งถ้ามีจุดที่ต้องการการแปลงเป็นจำนวนมาก ก็จะใช้เวลาในการคำนวณน้อยกว่าการคำนวณโดยวิธีแรก

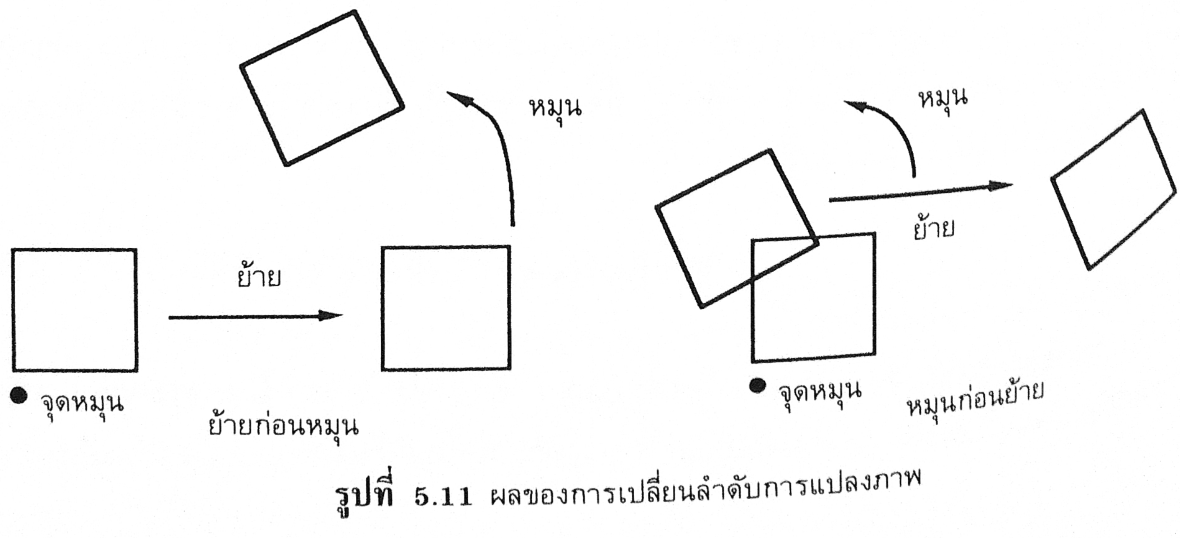
ลำดับของการแปลงภาพที่นำมาผสมกันจะให้ผลที่ต่างกัน เช่น การแปลงภาพโดยการหมุนภาพก่อนแล้วย้ายภาพ จะให้ผลต่างกันกับการย้ายภาพก่อนแล้วค่อยหมุนภาพ รูปที่ 5.11 แสดงผลภาพที่ได้เมื่อลำดับการแปลงภาพต่างกัน ซึ่งในการคำนวณโดยใช้แมทริกซ์ก็ต้องลำดับแมทริกซ์ของการแปลงแต่ละแบบให้ถูกต้อง จึงจะให้ผลของการแปลงได้ถูกต้อง

5.9 โปรแกรมย่อยสำหรับการแปลงภาพ

ในระบบกราฟิกส์จะเตรียมคำสั่งในการแปลงภาพให้ โดยที่การคำนวณทางแมทริกซ์ เช่น การคูณแมทริกซ์จะใช้ภาษาเครื่องซึ่งจะทำการคำนวณได้เร็วกว่า หรือไม่ก็ใช้ฮาร์ดแวร์พิเศษซึ่งจะช่วยให้การคำนวณเร็วมากขึ้น คำสั่งที่สำคัญมีดังนี้

Translate(Tx,Ty)	สำหรับย้ายภาพไปทางแกน x เป็นระยะ Tx หน่วยและทางแกน y เป็นระยะ Ty หน่วย
Rotate(θ,Xr,Yr)	**สำหรับหมุนภาพรอบจุด (Xr,Yr) เป็นมุม θ ทวนเข็มนาฬิกา θ เป็นมุมเรเดียน**
Scale(Sx,Sy,Xf,Yf)	ย่อหรือขยายภาพ Sx (ทางแกน x) เท่าของภาพเดิมและทางแกน y ย่อหรือขยาย Sy เท่าของภาพเดิม โดยมีจุด (Xf,Yf) เป็นจุดประจำที่

สำหรับการแปลงแบบอื่น ๆ เช่น การบิดภาพจะไม่ได้เตรียมไว้ในระบบกราฟิกส์ทั่วไป เนื่องจากไม่ค่อยได้ใช้งานมากนักจะใช้ในงานพิเศษบางอย่างเท่านั้น ดังนั้นในที่นี้ก็จะไม่กล่าวถึงเช่นกัน



รูปที่ 5.11 ผลของการเปลี่ยนลำดับการแปลงภาพ

ก่อนที่เราจะใช้คำสั่งเกี่ยวกับการแปลงภาพ เราควรทำความเข้าใจก่อนว่าการแปลงภาพนี้มีผลต่อคำสั่งเกี่ยวกับการวาดภาพดังนี้

ประการแรก จุดพิกัดที่นำมาแปลงนั้นจะเป็นได้ทั้งจุดพิกัดดั้งเดิมของภาพและจุดพิกัดซึ่งถูกแปลงมาแล้ว ตัวอย่างเช่น ถ้าเราต้องการย้ายภาพ แล้วหมุนภาพจุดพิกัดเดิมของภาพจะผ่านการแปลงแบบย้ายภาพ จากนั้นจุดพิกัดที่จะนำไปผ่านการแปลงแบบหมุนภาพก็คือจุดพิกัดซึ่งผ่านการแปลงแบบย้ายภาพมาแล้ว

ประการที่ 2 ถ้าเราแสดงภาพบนจอภาพแล้วสั่งให้มีการแปลงภาพแบบใดแบบหนึ่ง หลังจากสั่งแล้วจะไม่ปรากฏภาพที่ผ่านการแปลงภาพบนจอภาพให้ ถ้าต้องการให้แสดงภาพที่ผ่านการแปลงภาพจะต้องใช้คำสั่งให้นำพิกัดใหม่นั้นมาสร้างภาพเอง

ดังที่กล่าวมาแล้วข้างต้นว่า ระบบกราฟิกส์ทั่วไปจะเตรียมคำสั่งเกี่ยวกับการแปลงภาพไว้ให้แล้วโดยใช้ภาษาเครื่องหรือไม่ก็ฮาร์ดแวร์พิเศษ แต่ถ้าเราต้องการจะสร้างซอฟท์แวร์กราฟิกส์ไว้ใช้เอง เราก็จะต้องเขียนโปรแกรมสำหรับการแปลงภาพเอง ซึ่งในที่นี้เราจะใช้ภาษาซีแล้วใช้การคำนวณแบบแมทริกซ์ ดังนั้นเราก็จะมีคำสั่งเกี่ยวกับแมทริกซ์ เช่น การคูณแมทริกซ์ การทำให้แมทริกซ์เป็นแมทริกซ์ไอเดนติตี้ (identity matrix) เป็นต้น รวมเข้าไว้ในโปรแกรมของเราด้วย ซึ่งการนำแมทริกซ์การแปลง 2 แบบมาคูณกันก็คือ การแปลงภาพ 2 แบบนั่นเอง เช่นนำเอาแมทริกซ์การแปลงภาพแบบย้ายภาพคูณกับแมทริกซ์การแปลงภาพแบบหมุนภาพ หมายถึงว่า เราต้องการย้ายภาพก่อนแล้วค่อยหมุนภาพ ซึ่งต่างกันกับการหมุนภาพก่อนแล้วย้ายภาพดังได้กล่าวมาแล้วข้างต้น ส่วนการทำให้แมทริกซ์เป็นแมทริกซ์ไอเดนติตี้ ก็คือ การยกเลิกการแปลงภาพที่ทำผ่านมาแล้วทั้งหมด

โปรแกรมที่ 5.2 เป็นโปรแกรมแสดงการแปลงภาพแบบต่าง ๆ ผสมกัน โดยใช้การคำนวณแบบแมทริกซ์ ในโปรแกรมนี้จะเริ่มจากวาดภาพเดิมที่จุดกำเนิดก่อน จากนั้นจะทำการย้ายภาพไปที่จุด (300,100) แล้วขยายภาพ 2 เท่า โดยสัมพันธ์กับจุดประจำที่ (300,100) จากภาพขยายที่ได้ทำการย่อภาพลง 0.3 เท่า โดยจุดประจำที่อยู่ที่เดิม แล้วทำการหมุนภาพไป 135 องศา โดยจุดหมุนอยู่ที่จุด (300,100)

โปรแกรมที่ 5.2

```
/* A C program that demonstrates combining 2D transformations */

#include <graphics.h>
#include <math.h>
```

```
#define PI 3.14159
#define ROW 3
#define COL 3
#define PTS_NO 5

typedef struct coord {  /* structure of points which define image */
      int x;
      int y;
      int c;
};

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Out_Text_XY(int x,int y,char text[]);
void Clear_Viewport(void);

void Draw_Image(struct coord imagedef[PTS_NO]);
void Make_Identity(float m[ROW][COL]);
void Combine_Transformation(float t[ROW][COL],float m[ROW][COL]);
void Transform_Image(struct coord imagedef[PTS_NO],float t[ROW][COL]);
void Translate(int tx,int ty,float t[ROW][COL]);
void Scale(float sx,float sy,int xf,int yf,float t[ROW][COL]);
void Rotate(float degree,int xr,int yr,float t[ROW][COL]);

main()
{

  /* structure of points which define image  */
  struct coord image[PTS_NO] = { /* image definition */
        0,0,1,
        100,100,1,
```

```
    50,0,1,
    0,50,1,
    100,100,1
};
float t[ROW][COL];

Initialize_Graphics_Mode();

/* draw original image */
Draw_Image(image);
Out_Text_XY(5,110,"original image");
getchar();
Clear_Viewport();

/* translate the image (300,100) */
Make_Identity(t);
Translate(300,100,t);
Transform_Image(image,t);
Draw_Image(image);
Out_Text_XY(300,210,"translated image");
getchar();
Clear_Viewport();

/* double the image, fixed point = (300,100) */
Make_Identity(t);
Scale(2,2,300,100,t);
Transform_Image(image,t);
Draw_Image(image);
Out_Text_XY(300,300,"enlarged image");
getchar();
Clear_Viewport();

/* reduced the image, fixed point = (300,100) */
```

```
    Make_Identity(t);
    Scale(0.5,0.5,300,100,t);
    Transform_Image(image,t);
    Draw_Image(image);
    Out_Text_XY(300,200,"reduced image");
    getchar();
    Clear_Viewport();

    /* rotate image */
    Make_Identity(t);
    Rotate(135,300,100,t);
    Transform_Image(image,t);
    Draw_Image(image);
    Out_Text_XY(300,150,"rotated image");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
```

```
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Clear_Viewport(void)
{
    clearviewport();
}
```

```
void Draw_Image(struct coord imagedef[PTS_NO])
{
    int i;

    for(i=0; i<PTS_NO ; i++) {
        if(i==0) {
            Move_To(imagedef[i].x,imagedef[i].y);
        }
        Draw_Line_To(imagedef[i].x,imagedef[i].y);
    }
}

void Transform_Image(struct coord imagedef[PTS_NO],float t[ROW][COL])
{
    int k;
    float temp;

    for(k=0; k<PTS_NO; k++) {
        temp = imagedef[k].x*t[0][0]+imagedef[k].y*t[1][0]+t[2][0];
        imagedef[k].y = imagedef[k].x*t[0][1]
                        +imagedef[k].y*t[1][1]+t[2][1];
        imagedef[k].x = temp;
    }
}

void Combine_Transformation(float t[ROW][COL],float m[ROW][COL])
/* t = combined transformation matrix */
{
    int r, c;
    float temp[ROW][COL];
    for(r=0 ;r<ROW; r++)
        for(c=0 ;c<COL; c++)
```

```
            temp[r][c] = t[r][0]*m[0][c]+t[r][1]*m[1][c]
                        +t[r][2]*m[2][c];
    for(r=0 ;r<ROW; r++)
        for(c=0 ;c<COL; c++)
            t[r][c] = temp[r][c];
}

void Make_Identity(float m[ROW][COL])
{
    int r,c;

    for(r=0; r<ROW; r++)
        for(c=0; c<COL; c++)
            if (r==c) m[r][c] = 1; else m[r][c] = 0;
}

void Translate(int tx,int ty,float t[ROW][COL])
{
    float matrix[ROW][COL];

    Make_Identity(matrix);
    matrix[2][0] = tx; matrix[2][1] = ty;
    Combine_Transformation(t,matrix);
}

void Scale(float sx,float sy,int xf,int yf,float t[ROW][COL])
/* (xf,yf) = fixed point */
{
    float matrix[ROW][COL];
    Make_Identity(matrix);
    matrix[0][0] = sx; matrix[1][1] = sy;
    matrix[2][0] = (1-sx)*xf;
```

```
    matrix[2][1] = (1-sy)*yf;
    Combine_Transformation(t,matrix);
}

void Rotate(float degree,int xr,int yr,float t[ROW][COL])
/* (xr,yr) = pivot point */
{
    float matrix[ROW][COL];
    float cdg,sdg,radian;

    Make_Identity(matrix);
    radian = degree*PI/180;
    cdg = cos(radian); sdg = sin(radian);
    matrix[0][0] = cdg; matrix[0][1] = sdg;
    matrix[1][0] = -sdg; matrix[1][1] = cdg;
    matrix[2][0] = (1-cdg)*xr + yr*sdg;
    matrix[2][1] = (1-cdg)*yr - xr*sdg;
    Combine_Transformation(t,matrix);
}
```

5.10 การประยุกต์

โปรแกรมที่จะแสดงต่อไปนี้เป็นการนำเอาการแปลงภาพแบบหมุนมาใช้ เพื่อสร้างเป็นภาพใหม่ ๆ ที่น่าสนใจขึ้นได้

โปรแกรมที่ 5.3 เป็นโปรแกรมที่สร้างรูปสี่เหลี่ยมขึ้นแล้วทำการหมุนภาพนี้ไปเป็นมุมต่าง ๆ

```
/* A C program that demonstrates some picture */

#include <graphics.h>
#include <math.h>

#define PI 3.14159
```

```
#define ROW 3
#define COL 3
#define PTS_NO 5

typedef struct coord {   /* structure of points which define image */
      int x;
      int y;
      int c;
};

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Clear_Viewport(void);
void Draw_Image(struct coord imagedef[PTS_NO]);
void Make_Identity(float m[ROW][COL]);
void Combine_Transformation(float t[ROW][COL],float m[ROW][COL]);
void Transform_Image(struct coord imagedef[PTS_NO],float t[ROW][COL]);
void Rotate(float degree,int xr,int yr,float t[ROW][COL]);

main()
{
    /* structure of points which define image  */
    struct coord image[PTS_NO] = { /* image definition */
          200,200,1,
          350,200,1,
          350,250,1,
          200,250,1,
          200,200,1
    };
    float t[ROW][COL];
```

```
    int i,n;

    Initialize_Graphics_Mode();
    Clear_Viewport();

    n = 18;  /* number of rotations */
    for (i=1; i<=n; i++) {
            Draw_Image(image);
            Make_Identity(t);
            Rotate(360/n,275,225,t);
            Transform_Image(image,t);
    }

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
            printf("Graphics error : %s\n", grapherrormsg(errorcode));
            printf("Press any key to halt : ");
```

```
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Clear_Viewport(void)
{
    clearviewport();
}

void Draw_Image(struct coord imagedef[PTS_NO])
{
    int i;
    for(i=0; i<PTS_NO ; i++) {
        if(i==0) {
            Move_To(imagedef[i].x,imagedef[i].y);
        }
```

```
            Draw_Line_To(imagedef[i].x,imagedef[i].y);
        }
}

void Transform_Image(struct coord imagedef[PTS_NO],float t[ROW][COL])
{
    int k;
    float temp;

    for(k=0; k<PTS_NO; k++) {
            temp = imagedef[k].x*t[0][0]+imagedef[k].y*t[1][0]+t[2][0];
            imagedef[k].y = imagedef[k].x*t[0][1]
                            +imagedef[k].y*t[1][1]+t[2][1];
            imagedef[k].x = temp;
    }
}

void Combine_Transformation(float t[ROW][COL],float m[ROW][COL])
/* t = combined transformation matrix */
{
    int r, c;
    float temp[ROW][COL];

    for(r=0 ;r<ROW; r++)
          for(c=0 ;c<COL; c++)
              temp[r][c] = t[r][0]*m[0][c]+t[r][1]*m[1][c]
                              +t[r][2]*m[2][c];
    for(r=0 ;r<ROW; r++)
          for(c=0 ;c<COL; c++)
              t[r][c] = temp[r][c];
}

void Make_Identity(float m[ROW][COL])
```

```
{
    int r,c;

    for(r=0; r<ROW; r++)
          for(c=0; c<COL; c++)
               if (r==c) m[r][c] = 1; else m[r][c] = 0;
}

void Rotate(float degree,int xr,int yr,float t[ROW][COL])
/* (xr,yr) = pivot point */
{
    float matrix[ROW][COL];
    float cdg,sdg,radian;

    Make_Identity(matrix);
    radian = degree*PI/180;
    cdg = cos(radian); sdg = sin(radian);
    matrix[0][0] = cdg; matrix[0][1] = sdg;
    matrix[1][0] = -sdg; matrix[1][1] = cdg;
    matrix[2][0] = (1-cdg)*xr + yr*sdg;
    matrix[2][1] = (1-cdg)*yr - xr*sdg;
    Combine_Transformation(t,matrix);
}
```

โปรแกรมที่ 5.4 เป็นโปรแกรมแสดงการแปลงภาพแบบหมุนภาพที่ใช้เป็นภาพเครื่องบินที่บินหมุนไปบนจอ

```
/* A C program that demonstrates some simple moving picture */

#include <graphics.h>
#include <math.h>

#define PI 3.14159
```

```
#define ROW 3
#define COL 3
#define PTS_NO 20

typedef struct coord {  /* structure of points which define image */
        int x;
        int y;
        int c;
};

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Clear_Viewport(void);
void Draw_Image(struct coord imagedef[PTS_NO]);
void Make_Identity(float m[ROW][COL]);
void Combine_Transformation(float t[ROW][COL],float m[ROW][COL]);
void Transform_Image(struct coord imagedef[PTS_NO],float t[ROW][COL]);
void Rotate(float degree,int xr,int yr,float t[ROW][COL]);

main()
{
    /* structure of points which define image  */
    struct coord image[PTS_NO] = { /* airplane definition */
            200,200,1, 195,190,1, 195,175,1, 175,155,1, 175,145,1,
            195,150,1, 195,140,1, 185,135,1, 185,130,1, 195,135,1,
            205,135,1, 215,130,1, 215,135,1, 205,140,1, 205,150,1,
            225,145,1, 225,155,1, 205,175,1, 205,190,1, 200,200,1
    };
    float t[ROW][COL];
    int x = 200; /* y = 200, initial plane position */
```

```
    int theta = -20; /* initial rotation angle */

    Initialize_Graphics_Mode();

    while (x < 400) {
            Clear_Viewport();
            Draw_Image(image);
            x += 5; /* spiral effect */
            Make_Identity(t);
            Rotate(theta,150+x,200,t);
            Transform_Image(image,t);
    }

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
            printf("Graphics error : %s\n", grapherrormsg(errorcode));
            printf("Press any key to halt : ");
```

```
            getch();
            exit(1); /* return with error code */
        }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Clear_Viewport(void)
{
    clearviewport();
}

void Draw_Image(struct coord imagedef[PTS_NO])
{
    int i;
    for(i=0; i<PTS_NO ; i++) {
          if(i==0) {
              Move_To(imagedef[i].x,imagedef[i].y);
          }
```

```
        Draw_Line_To(imagedef[i].x,imagedef[i].y);
    }
}

void Transform_Image(struct coord imagedef[PTS_NO],float t[ROW][COL])
{
    int k;
    float temp;

    for(k=0; k<PTS_NO; k++) {
          temp = imagedef[k].x*t[0][0]+imagedef[k].y*t[1][0]+t[2][0];
          imagedef[k].y = imagedef[k].x*t[0][1]
                          +imagedef[k].y*t[1][1]+t[2][1];
          imagedef[k].x = temp;
    }
}

void Combine_Transformation(float t[ROW][COL],float m[ROW][COL])
/* t = combined transformation matrix */
{
    int r, c;
    float temp[ROW][COL];

    for(r=0 ;r<ROW; r++)
         for(c=0 ;c<COL; c++)
             temp[r][c] = t[r][0]*m[0][c]+t[r][1]*m[1][c]
                          +t[r][2]*m[2][c];
    for(r=0 ;r<ROW; r++)
         for(c=0 ;c<COL; c++)
             t[r][c] = temp[r][c];
}

void Make_Identity(float m[ROW][COL])
```

```
{
    int r,c;

    for(r=0; r<ROW; r++)
          for(c=0; c<COL; c++)
              if (r==c) m[r][c] = 1; else m[r][c] = 0;
}

void Rotate(float degree,int xr,int yr,float t[ROW][COL])
/* (xr,yr) = pivot point */
{
    float matrix[ROW][COL];
    float cdg,sdg,radian;

    Make_Identity(matrix);
    radian = degree*PI/180;
    cdg = cos(radian); sdg = sin(radian);
    matrix[0][0] = cdg; matrix[0][1] = sdg;
    matrix[1][0] = -sdg; matrix[1][1] = cdg;
    matrix[2][0] = (1-cdg)*xr + yr*sdg;
    matrix[2][1] = (1-cdg)*yr - xr*sdg;
    Combine_Transformation(t,matrix);
}
```

5.11 สรุป

ภาพเรขาคณิตต่าง ๆ อาจจะนิยามได้ด้วยลำดับของจุดพิกัดชุดหนึ่ง และการแปลงภาพแบบต่าง ๆ นั้นทำได้โดยการแปลงค่าจุดพิกัดที่นิยามภาพนั้น ๆ ให้เป็นลำดับของจุดพิกัดชุดใหม่ ซึ่งผ่านการแปลงแล้ว ภาพที่ได้ก็จะเป็นภาพเดิมแต่การแสดงภาพจะเป็นไปตามการแปลงในแบบที่ผู้ใช้ต้องการ

การแปลงภาพหลาย ๆ แบบผสมกันนั้น ทำให้เกิดการแสดงภาพที่แตกต่างกันไปได้มากมาย การใช้แมทริกซ์มาแทนลักษณะการแปลงแบบต่าง ๆ จะทำให้การแปลงภาพทำได้ง่ายและสะดวกขึ้น

6 เซกเมนต์

6.1 บทนำ

ภาพที่มีความซับซ้อนมักจะประกอบไปด้วยภาพย่อยง่าย ๆ หลาย ๆ ภาพมาประกอบกัน ภาพย่อยบางส่วนอาจจะเป็นภาพย่อยซึ่งมีอยู่แล้วแต่ถูกแปลงภาพ โดยการย้ายภาพ การหมุนภาพ การย่อขยายภาพ หรือการบิดภาพ ให้กลายเป็นภาพย่อยภาพใหม่ ตัวอย่างเช่น ภาพเหตุการณ์ต่าง ๆ บนถนน จะประกอบด้วยรถแบบต่าง ๆ ขนาดต่าง ๆ ในตำแหน่งต่าง ๆ และทิศทางการวิ่งของรถก็ต่าง ๆ กัน รถอาจจะเป็นแบบเดียวกันแต่วิ่งในทิศทางต่างกัน หรือมีขนาดต่างกันตามระยะห่างของการมองเห็น นอกจากนี้ยังมีส่วนประกอบอื่น ๆ คือ ตึกหรือบ้านริมถนน ป้ายสัญลักษณ์การจราจร คนเดินบนทางเท้า เป็นต้น ภาพที่กล่าวมาข้างต้นเป็นภาพเหตุการณ์ที่เกิดขึ้นในเวลาหนึ่ง ถ้าเวลาผ่านไปภาพย่อยต่าง ๆ ก็จะมีการเปลี่ยนตำแหน่งไป เช่น รถก็ควรจะวิ่งไป คนเดินบนทางเท้าก็ควรมีการเปลี่ยนตำแหน่ง ส่วนตึก บ้าน และ ป้ายสัญลักษณ์การจราจรก็ควรจะอยู่กับที่ ดังนั้นในระบบกราฟิกส์จึงควรมีความสามารถที่จะเปลี่ยนตำแหน่งของภาพย่อยบางส่วนได้โดยที่ภาพย่อยส่วนอื่น ๆ ยังคงอยู่กับที่

วิธีการที่จะทำให้ระบบกราฟิกส์มีความสามารถเช่นนี้ก็คือ เราจะต้องสามารถแบ่งเก็บภาพเป็นส่วนย่อย ๆ ได้โดยแบ่งเก็บเป็นเซกเมนต์ (segment) ของแต่ละภาพย่อย ภาพย่อยแต่ละส่วนก็จะเป็นอิสระต่อกัน นั่นคือเราสามารถควบคุมภาพย่อยแต่ละส่วนได้โดยไม่กระทบกับภาพย่อยส่วนอื่น ๆ ภาพย่อยซึ่งเก็บในรูปของเซกเมนต์ก็คือ การนำลำดับคำสั่งทางกราฟิกส์มาเรียงต่อกันเพื่อสร้างให้เป็นภาพย่อยนั้นแล้วนำคำสั่งทั้งหมดนี้มารวมกัน โดยกำหนดชื่ออ้างอิงชื่อหนึ่งซึ่งจะถือเสมือนว่าเป็นคำสั่งเพียง 1 คำสั่ง เมื่อเราทำภาพย่อยให้เป็นเซกเมนต์แล้ว เราก็จะสามารถจัดการลบ

ภาพย่อยนั้นหรือแปลงภาพย่อยนั้นในแบบต่าง ๆ ตามที่เราต้องการได้โดยง่าย การทำให้เป็นเซกเมนต์นี้ก็คล้าย ๆ กับการแบ่งโปรแกรมออกเป็นโปรแกรมย่อยนั่นเอง การเรียกเซกเมนต์มาแสดงก็เหมือน ๆ กับการเรียกโปรแกรมย่อยมาใช้นั่นเอง

6.2 ไฟล์แสดงภาพ

ดังที่เราทราบมาแล้วว่าภาพย่อยสร้างขึ้นมาได้จากการนำคำสั่งทางกราฟิกส์มาลำดับต่อ ๆ กันแล้วรวมลำดับคำสั่งเหล่านั้น ซึ่งใช้แทนภาพย่อยเข้าไว้ในไฟล์แสดงภาพ (display file) สำหรับคำสั่งที่เก็บในไฟล์แสดงภาพ เราจะใช้ข้อมูลแบบโครงสร้างสำหรับแทนคำสั่งเหล่านี้ โดยจะประกอบไปด้วย 3 ฟิลด์ (field : เขตข้อมูล) ฟิลด์หนึ่งสำหรับชื่อคำสั่งกราฟิกส์ส่วนอีก 2 ฟิลด์ใช้สำหรับแทนพารามิเตอร์ซึ่งคำสั่งนั้น ๆ ต้องการ และไฟล์แสดงภาพก็จะเป็นอาร์เรย์ของโครงสร้างดังกล่าว (รูปที่ 6.1)

	Commands	Operand 1	Operand 2
1	MoveTo	100	100
2	DrawLineTo	200	50
3			
4			
• • •			
SIZE			

รูปที่ 6.1 ไฟล์แสดงภาพ

โครงสร้างข้อมูลของไฟล์แสดงภาพนอกเหนือจากการใช้อาร์เรย์แล้ว ก็คือ โครงสร้างแบบรายการเชื่อมโยงหรือลิงก์ลิสต์ (link lists) ซึ่งเป็นโครงสร้างที่เราไม่ต้องกำหนดขนาดให้ก่อนแต่เราสามารถเพิ่มหรือลดขนาดของมันได้ ซึ่งก็เหมาะสมที่จะใช้เป็นโครงสร้างข้อมูลของไฟล์แสดงภาพเนื่องจากเรามักจะไม่ทราบก่อนว่าจะมีจำนวนคำสั่งเท่าไรในไฟล์แสดงภาพ แต่การจัดการกับลิงก์ลิสต์ เช่น การค้นหาในลิงก์ลิสต์จะใช้เวลามากและเราต้องใช้พอยน์เตอร์ในการสร้างลิงก์ลิสต์ซึ่งทำให้เกิดความยุ่งยากซับซ้อนในการเขียนโปรแกรมเกินไป ดังนั้นในที่นี้เราจึงเลือกโครงสร้างอาร์เรย์เป็นโครงสร้างข้อมูลของไฟล์แสดงภาพ

โปรแกรมที่ 6.1 เป็นโปรแกรมตัวอย่างการสร้างไฟล์แสดงภาพ ซึ่งเก็บข้อมูลเป็นอาร์เรย์ของโครงสร้าง โปรแกรมจะอ่านข้อมูลคำสั่งและพารามิเตอร์จากไฟล์ นำไปเก็บในไฟล์แสดงภาพแล้วนำข้อมูลนั้นทั้งหมดมาแสดงเป็นภาพบนจอภาพ

โปรแกรมที่ 6.1

```
/* A C program that demonstrates how to construct display file
   in graphics system */

#include <graphics.h>
#include <stdio.h>

#define SIZE 500     /* maximum length of display file */

typedef enum commands {   /* user defined drawing commands */
        Stop = 0,
        MoveTo,
        MoveRel,
        DrawLineTo,
        DrawLineRel,
};

typedef struct entry {   /* display file entry */
        enum commands cmd;   /* graphics commands */
        int operand1;        /* 1st operand */
        int operand2;        /* 2st operand */
};

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Draw_Line_Rel(int dx,int dy);
```

```
void Move_Rel(int dx,int dy);
void Out_Text_XY(int x,int y,char text[]);
void Clear_Viewport(void);

void Get_Command(void);
void Draw_Image(int coms,int oper1,int oper2);


/* display file data structure */
struct entry display_file[SIZE];

int free_seg;

main()
{
    int i;

    Initialize_Graphics_Mode();
    free_seg = 1; /* set first position in display file */
    /* read commands from file and keep in display file */
    Get_Command();

    Clear_Viewport();
    for(i=1; i<free_seg; i++) {
          Draw_Image(display_file[i].cmd,
                     display_file[i].operand1,
                     display_file[i].operand2);
    };

    getchar();
    Close_Graphics_Mode();
    return(0);
}
```

```
void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
```

```
    moveto(x,y);
}

void Draw_Line_Rel(int dx,int dy)
{
    linerel(dx,dy);
}

void Move_Rel(int dx,int dy)
{
    moverel(dx,dy);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Clear_Viewport(void)
{
    clearviewport();
}

void Get_Command(void)
{
    int com;          /* commands code */
    int op1,op2;      /* first and second operand */
    FILE    *fin;

    fin = fopen("grcom.dat","r");  /* list of commands file */
    while (!feof(fin)) {
```

```
        fscanf(fin,"%d",&com);
        display_file[free_seg].cmd = com;
        fscanf(fin,"%d",&op1);
        display_file[free_seg].operand1 = op1;
        fscanf(fin,"%d",&op2);
        display_file[free_seg].operand2 = op2;
        free_seg++; /* next available position in display file */
    }
}

void Draw_Image(int coms,int oper1,int oper2)
{
    switch (coms) {
        case 0 : break;
        case 1 : Move_To(oper1,oper2);
                 break;
        case 2 : Move_Rel(oper1,oper2);
                 break;
        case 3 : Draw_Line_To(oper1,oper2);
                 break;
        case 4 : Draw_Line_Rel(oper1,oper2);
                 break;
    }
}
```

ฟังก์ชัน Get_Command จะอ่านลำดับของคำสั่งต่าง ๆ ซึ่งเก็บไว้ในไฟล์ชื่อ grcom.dat ซึ่งข้อมูลในไฟล์ grcom.dat เป็นดังนี้

```
1 10 10
3 100 10
3 100 100
3 10 100
```

```
3 10 10
5 95 95
0 0 0
1 200 100
3 300 100
3 200 200
3 200 100
0 0 0
1 150 250
3 300 250
1 150 250
3 150 300
1 150 250
3 150 200
1 150 250
3 100 250
0 0 0
1 300 300
4 50 0
4 25 -50
4 25 50
4 50 0
0 0 0
```

ตัวเลขตัวแรกในแถวก็คือรหัสของคำสั่ง ตัวที่ 2 และ 3 เป็นพารามิเตอร์ของคำสั่งนั้น ตัวอย่างเช่น 1 150 250 หมายถึงให้ย้ายจุด CP ไปยังตำแหน่งพิกัด (150,250) คำสั่งย้ายจุดก็คือคำสั่ง MoveTo (ซึ่งถ้าแปลงเป็นคำสั่งที่ใช้งานจริงในโปรแกรมก็คือฟังก์ชัน Move_To) รหัสคำสั่งกำหนดจากการกำหนดชนิดข้อมูลดังนี้

```
typedef enum commands {  /* user defined drawing commands */
     Stop = 0,
     MoveTo,
     MoveRel,
     DrawLineTo,
     DrawLineRel,
};
```

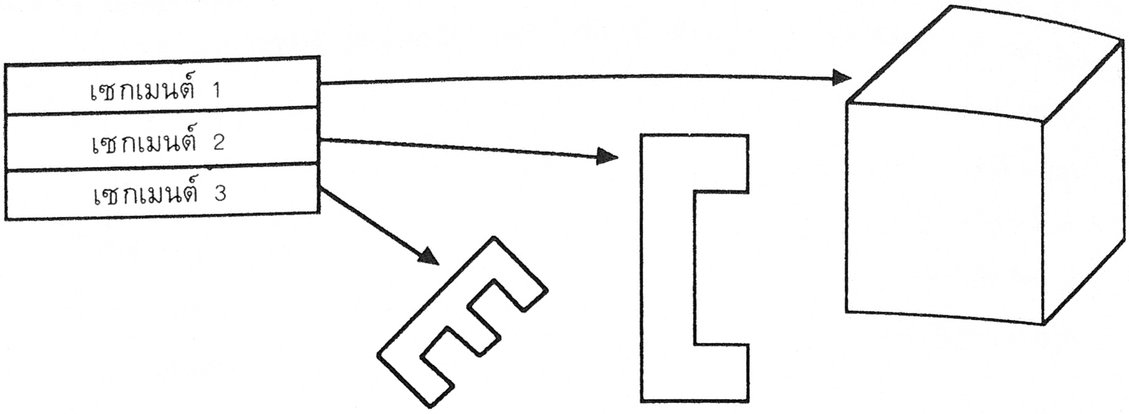
ซึ่งหมายความว่า คำสั่ง Stop มีรหัสเป็น 0 คำสั่ง MoveTo มีรหัสเป็น 1 ส่วนคำสั่งอื่น ๆ ก็มีรหัสเรียงกันไปตามลำดับ

โครงสร้างข้อมูลของไฟล์แสดงภาพคืออาร์เรย์ของโครงสร้างต่อไปนี้

```
typedef struct entry {     /* display file entry */
     enum commands cmd;    /* graphics commands */
     int operand1;         /* 1st operand */
     int operand2;         /* 2st operand */
};
/* display file data structure */
struct entry display_file[SIZE];
```

6.3 เซกเมนต์ในไฟล์แสดงภาพ

เนื่องจากไฟล์แสดงภาพประกอบไปด้วยคำสั่งที่ใช้สำหรับสร้างภาพที่ต้องการ การแบ่งไฟล์แสดงภาพออกเป็นส่วนย่อย ๆ เรียกว่า *เซกเมนต์* ซึ่งก็คือการแบ่งภาพออกเป็นส่วน ๆ แต่ละเซกเมนต์อาจจะถือเป็นเพียง 1 คำสั่งที่ใช้สร้างภาพย่อย 1 ภาพ ซึ่งเราสามารถจัดการย้ายภาพ หมุนภาพ ย่อขยายภาพ ไม่แสดงภาพ หรือการแปลงอื่น ๆ ได้โดยไม่มีผลกระทบต่อเซกเมนต์อื่น ๆ เลย (รูปที่ 6.2) (นอกจากมีการย้ายภาพให้ไปทับกับภาพอีกภาพหนึ่ง แต่เซกเมนต์ที่ใช้สร้างภาพแต่ละส่วนนั้นยังคงไม่เกี่ยวข้องกัน)



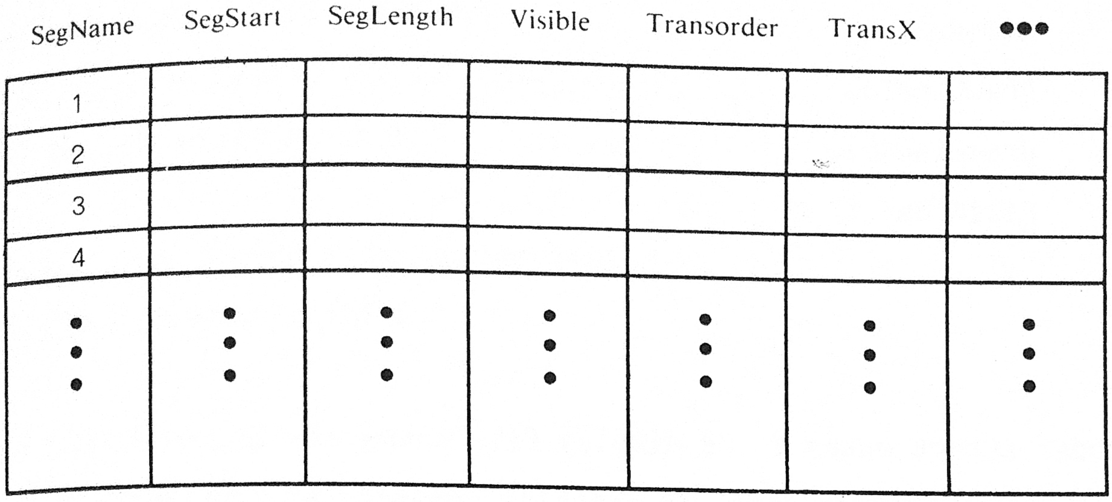
รูปที่ 6.2 เซกเมนต์ในไฟล์แสดงภาพกับภาพที่สร้างจากแต่ละเซกเมนต์

ในการแสดงภาพย่อยโดยใช้เซกเมนต์นั้น เราต้องกำหนดคำสั่งโดยมีแอตทริบิวต์รวมอยู่ด้วย แอตทริบิวต์ตัวหนึ่งอาจจะใช้ในการแสดงภาพตรงตำแหน่งที่กำหนดโดยจัดการย้ายภาพ แอตทริบิวต์อีกตัวหนึ่งใช้แสดงว่าต้องการหมุนภาพ และแอตทริบิวต์อื่น ๆ ก็ใช้แสดงการแปลงภาพอื่น ๆ ภาพย่อยอาจจะถูกลบไปได้โดยการเปลี่ยนแอตทริบิวต์การแสดงภาพของเซกเมนต์นั้นให้เป็นไม่แสดงภาพ ภาพย่อยก็จะหายไป หรือถ้าเปลี่ยนแอตทริบิวต์ให้เป็นต้องการแสดงภาพก็จะได้ภาพย่อยนั้นแสดงออกมา เนื่องจากลำดับของการแปลงภาพมีผลต่อรูปร่างและตำแหน่งของภาพรวมที่ได้ ดังนั้นจึงต้องมีแอตทริบิวต์สำหรับกำหนดลำดับการแปลงด้วย

เซกเมนต์ 1 เซกเมนต์อาจจะใช้แสดงภาพเดียวกันโดยใช้แอตทริบิวต์ต่างกันได้ เช่น เรามีเซกเมนต์สำหรับแสดงภาพอักษร A เราสามารถแสดงภาพย่อย A ในส่วนหนึ่งของจอภาพ แล้วแสดงภาพอักษร A ในอีกส่วนหนึ่งโดยย่อขนาดลง หรือหมุนภาพไป โดยใช้เซกเมนต์เดิมแต่เปลี่ยนแอตทริบิวต์ให้มีการย่อภาพหรือการหมุนภาพและสามารถแสดงภาพอักษร A ในส่วนอื่น ๆ ได้อีก โดยที่เรามีเซกเมนต์สำหรับแสดงภาพอักษร A เพียงเซกเมนต์เดียวเท่านั้น และการแสดงภาพหลาย ๆ แบบนี้ไม่ทำให้เซกเมนต์สำหรับแสดงภาพอักษร A ดั้งเดิมเสียไป

6.4 ตารางเซกเมนต์

ไฟล์แสดงภาพประกอบไปด้วยเซกเมนต์จำนวนมาก ในการจะอ้างอิงเซกเมนต์ได้ถูกต้อง เราต้องกำหนดชื่อเฉพาะให้แต่ละเซกเมนต์และรู้ตำแหน่งของเซกเมนต์ในไฟล์แสดงภาพ กล่าวคือ รู้ว่าเซกเมนต์ที่ต้องการค้นหานั้นเริ่มที่ตำแหน่งใดและสิ้นสุดที่ตำแหน่งใดในไฟล์แสดงภาพ ดังนั้นในเซกเมนต์จะมีแอตทริบิวต์สำหรับใช้บอกตำแหน่งเริ่มต้นในไฟล์แสดงภาพและความยาวของเซกเมนต์ (เพื่อบอกจุดสิ้นสุดของเซกเมนต์) และเพื่อให้สะดวกในการจัดการ เราก็จะเก็บชื่อของเซกเมนต์และแอตทริบิวต์ของเซกเมนต์ในโครงสร้างข้อมูลที่เรียกว่า *ตารางเซกเมนต์ (segment table)* ซึ่งช่วยให้เราสามารถอ้างอิงและจัดการแต่ละเซกเมนต์ได้สะดวกมากขึ้น (รูปที่ 6.3)



รูปที่ 6.3 ตารางเซกเมนต์

ข้อมูลแบบโครงสร้างซึ่งมีฟิลด์ต่าง ๆ เป็นแอตทริบิวต์จะนำมาใช้เป็นโครงสร้างข้อมูลของเซกเมนต์ ส่วนตารางเซกเมนต์จะเป็นอาร์เรย์ของโครงสร้างดังกล่าว ถ้าเราใช้เลขจำนวนเต็มบวกเป็นชื่อเฉพาะของเซกเมนต์ ตัวเลขนี้ก็สามารถใช้เป็นตัวชี้ของอาร์เรย์ได้

โปรแกรมที่ 6.2 เป็นโปรแกรมสำหรับจัดการแบ่งไฟล์แสดงภาพออกเป็นเซกเมนต์ โดยใช้รหัส 0 ซึ่งก็คือคำสั่ง Stop เป็นตัวแยกเซกเมนต์ สำหรับไฟล์ grcom.dat จะมีอยู่ทั้งหมด 4 เซกเมนต์

โปรแกรมที่ 6.2

```
/* A C program that demonstrates how to construct
   display file segments in graphics system */

#include <graphics.h>
#include <stdio.h>

#define SIZE 500     /* maximum length of display file */
#define MAX_SEG 50   /* maximum number of segments */

typedef enum commands {   /* user defined drawing commands */
        Stop = 0,
        MoveTo,
```

```
        MoveRel,
        DrawLineTo,
        DrawLineRel,
        FillArea

};

typedef struct entry {  /* display file entry */
        enum commands cmd;  /* graphics commands */
        int operand1;       /* 1st operand */
        int operand2;       /* 2st operand */
};

typedef struct table_entry { /* entry in segment table */
int seg_start;   /* start position of a segment
                    in display file */
int seg_length; /* length of a segment
                    (assign the end position) */
int visible;     /* display, if visible ≠ 1 */
int transorder; /* transformation order */
int trans_x,
    trans_y;     /* translation */
int scale_x,
    scale_y;     /* scaling */
int rot_angle;   /* rotation */
};

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);

void Move_To(int x,int y);
void Move_Rel(int dx,int dy);
```

```
void Draw_Line_To(int x,int y);
void Draw_Line_Rel(int dx,int dy);
void Fill_Area(int x,int y);

void Out_Text_XY(int x,int y,char text[]);
void Clear_Viewport(void);

void Read_Make_Seg(void);
void Get_Command(void);
void Create_Seg(int number);
void Close_Seg(int number);
void Make_Seg(int number);
void Convert_To_Image(int coms,int oper1,int oper2);
void Display_Seg(int number);

/* display file data structure */
struct entry display_file[SIZE];

/* segment table data structure */
struct table_entry seg_table[MAX_SEG];

/* global variables */
int seg_open = 0; /* seg_open = 1 if a segment is being constructed */
int next_entry;    /* index of next entry */
int cmd_index = 0;/* index of next available entry in display file */
int error = 0;     /* error = 1 if an error has occured */
int num_seg = 1;   /* segment counter */

main()
{
    Initialize_Graphics_Mode();

     /* read and make segment */
```

```
    Read_Make_Seg();

    /* display image of each segment */
    Display_Seg(1);   /* display segment no. 1 */
    Display_Seg(2);
    Display_Seg(3);
    Display_Seg(4);
    getchar();

    Display_Seg(5);  /* display invalid segment */

    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");
    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
```

```
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Move_Rel(int dx,int dy)
{
    moverel(dx,dy);
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}
void Draw_Line_Rel(int dx,int dy)
{
    linerel(dx,dy);
}

void Fill_Area(int x,int y)
{
    floodfill(x,y,15);
}

void Out_Text_XY(int x,int y,char text[])
```

```
{
    outtextxy(x,y,text);
}

void Clear_Viewport(void)
{
    clearviewport();
}

void Get_Command(void)
{
    int com;         /* commands code */
    int op1,op2;     /* first and second operand */
    FILE    *fin;

    fin = fopen("grcom.dat","r");  /* list of commands file */
    while (!feof(fin)) {
          cmd_index++; /* next available position in display file */
          fscanf(fin,"%d",&com);
          display_file[cmd_index].cmd = com;
          fscanf(fin,"%d",&op1);
          display_file[cmd_index].operand1 = op1;
          fscanf(fin,"%d",&op2);
          display_file[cmd_index].operand2 = op2;
    }
}

void Create_Seg(int number)
{
    if(seg_open==1) {
           printf("Error! : Another segment is open");
           error = 1;
```

```
        exit(1); /* exit Create_Seg */
    }
    if((number<1) || (number>MAX_SEG)) {
        printf("Error! : Invalid segment number");
        error = 1;
        exit(1); /* exit Create_Seg */
    }
    if(seg_table[number].seg_length>0) {
        printf("Error! : Segment already created");
        error = 1;
        exit(1); /* exit Create_Seg */
    }
    /* starting position in display file for segment */
    seg_table[number].seg_start = next_entry;
    /* set default attributes values */
    seg_table[number].visible = 1;      /* display */
    seg_table[number].transorder = 1;
    seg_table[number].trans_x = 0;      /* no translation */
    seg_table[number].trans_y = 0;
    seg_table[number].scale_x = 1;      /* no scaling */
    seg_table[number].scale_y = 1;
    seg_table[number].rot_angle = 0;    /* no rotation */
    seg_open = 1;
}

void Close_Seg(int number)
{
    if(!seg_open) {
        printf("Error! : No segment to be closed");
        error = 1;
        exit(1);
    }
```

```
    seg_table[number].seg_length
            = next_entry - seg_table[number].seg_start;
    seg_open = 0;
}

void Make_Seg(int number)
{
    Create_Seg(number);
    next_entry++;
    while (display_file[next_entry].cmd != 0)
            next_entry++;
    Close_Seg(number);
    num_seg++;  /* count segment */
}

void Read_Make_Seg(void)
{
    /* read commands from file into display file */
    Get_Command();

    /* make segment table from display file */
    for(next_entry=0; next_entry<=cmd_index; next_entry++) {
          Make_Seg(num_seg);
    }
    num_seg -= 1;    /* set the correct number of segment */
}

void Convert_To_Image(int coms,int oper1,int oper2)
{
    switch (coms) {
          case 0 : break;
          case 1 : Move_To(oper1,oper2);
```

```
                    break;
            case 2 : Move_Rel(oper1,oper2);
                     break;
            case 3 : Draw_Line_To(oper1,oper2);
                     break;
            case 4 : Draw_Line_Rel(oper1,oper2);
                     break;
            case 5 : Fill_Area(oper1,oper2);
                     break;
    }
}
void Display_Seg(int number)
{
    int i;
    if((number<1) || (number>num_seg)) {
        printf("Error! : Invalid segment number");
        error = 1;
        exit(1); /* exit Create_Seg */
    }

    for(i=seg_table[number].seg_start;
       i<(seg_table[number].seg_length+seg_table[number].seg_start);
       i++)
         Convert_To_Image(display_file[i].cmd,
                          display_file[i].operand1,
                          display_file[i].operand2);
}
```

โปรแกรมจะอ่านไฟล์คำสั่งจาก grcom.dat ไปเก็บไว้ในไฟล์แสดงภาพ แล้วจัดการแบ่งให้เป็นเซกเมนต์โดยใช้ฟังก์ชัน Create_Seg Make_Seg และ Close_Seg แล้วเก็บข้อมูลเซกเมนต์ไว้ในตารางเซกเมนต์ ซึ่งมีโครงสร้างข้อมูลดังนี้

```
typedef struct table_entry { /* entry in segment table */
      int seg_start;  /* start position of a segment
                         in display file */
      int seg_length; /* length of a segment
                         (assign the end position) */
      int visible;    /* display, if visible = 1 */
      int transorder; /* transformation order */
      int trans_x,
          trans_y;    /* translation */
      int scale_x,
          scale_y;    /* scaling */
      int rot_angle;  /* rotation */
};

/* segment table data structure */
struct table_entry seg_table[MAX_SEG];
```

6.5 การดำเนินการกับเซกเมนต์ในไฟล์แสดงภาพ

ในการดำเนินการกับเซกเมนต์นั้นมีฟังก์ชันที่นำมาใช้งานหลายฟังก์ชันซึ่งได้แก่ ฟังก์ชันที่เกี่ยวกับการสร้างเซกเมนต์ การกำหนดแอตทริบิวต์ให้กับเซกเมนต์ และการแสดงเซกเมนต์บนจอภาพ ในตอนนี้จะกล่าวถึงการทำงานของแต่ละฟังก์ชันอย่างคร่าว ๆ สำหรับรายละเอียดของแต่ฟังก์ชันสามารถศึกษาได้จากตัวอย่างโปรแกรมที่ 6.3

ก่อนจะกล่าวถึงการทำงานของแต่ละฟังก์ชัน เราจะกำหนดตัวแปรซึ่งทุกฟังก์ชันจะรู้จักและใช้ร่วมกัน (global variables) ดังนี้

```
/* global variables */
int seg_open = 0; /* seg_open = 1 if a segment is being constructed */
int next_entry;   /* index of next entry */
int cmd_index = 0;/* index of next available entry in display file */
```

```
int error = 0;      /* error = 1 if an error has occured */
int num_seg = 1;    /* segment counter */

float t[ROW][COL];/* transformed matrix */
```

โปรแกรมที่ 6.3 ตัวอย่างการแสดงภาพแต่ละเซกเมนต์ซึ่งถูกกำหนดแอตทริบิวต์ต่าง ๆ กันไป

```
/* A C program that demonstrates how to set transformation
   attributes to the segments */

#include <graphics.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define PI 3.14159
#define ROW 3
#define COL 3
#define SIZE 500      /* maximum length of display file */
#define MAX_SEG 50    /* maximum number of segments */

typedef enum commands {   /* user defined drawing commands */
     Stop = 0,
     MoveTo,
     MoveRel,
     DrawLineTo,
     DrawLineRel,
     FillArea
};

typedef struct entry {   /* display file entry */
     enum commands cmd;   /* graphics commands */
```

```
        int operand1;          /* 1st operand */
        int operand2;          /* 2st operand */
};

typedef struct table_entry { /* entry in segment table */
        int seg_start;  /* start position of a segment
                           in display file */
        int seg_length; /* length of a segment
                           (assign the end position) */
        int visible;    /* display, if visible = 1 */
        int transorder; /* transformation order
                           0    : no transformation
                           1    : translate
                           2    : scale
                           3    : rotate
                           12   : translate scale
                           13   : translate rotate   */
        int trans_x,
            trans_y;    /* translation */
        int scale_x,
            scale_y,    /* scaling parameters */
            fix_x,
            fix_y;      /* scaling fixed point */
        int rot_angle,  /* rotation angle */
            rot_x,
            rot_y;/* pivot point */
};

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);

void Move_To(int x,int y);
```

```
void Move_Rel(int dx,int dy);
void Draw_Line_To(int x,int y);
void Draw_Line_Rel(int dx,int dy);
void Fill_Area(int x,int y);

void Out_Text_XY(int x,int y,char text[]);
void Clear_Viewport(void);

void Read_Make_Seg(void);
void Get_Command(void);
void Create_Seg(int number);
void Close_Seg(int number);
void Make_Seg(int number);
void Convert_To_Image(int coms,int oper1,int oper2);
void Display_Seg(int number);

void Set_Transform(int number,          /* segment number */
                   int tx,int ty,   /* translation parameters */
                   int sx,int sy,   /* scaling parameters */
                   int fx,int fy,   /* fixed point */
                   int rot,           /* rotation angle */
                   int rx,int ry); /* pivot point */
void Set_Order(int number,int order);
void Set_Visible(int number,int see);
void Construct_Transform(int number);

void Make_Identity(float m[ROW][COL]);
void Combine_Transformation(float t[ROW][COL],float m[ROW][COL]);
void Transform_Image(int number,struct entry display_file[SIZE],
                     float t[ROW][COL]);
void Translate(int tx,int ty,float t[ROW][COL]);
void Scale(float sx,float sy,int xf,int yf,float t[ROW][COL]);
void Rotate(float degree,int xr,int yr,float t[ROW][COL]);
```

```
/* display file data structure */
struct entry display_file[SIZE];

/* segment table data structure */
struct table_entry seg_table[MAX_SEG];

/* global variables */
int seg_open = 0; /* seg_open = 1 if a segment is being constructed */
int next_entry;    /* index of next entry */
int cmd_index = 0;/* index of next available entry in display file */
int error = 0;     /* error = 1 if an error has occured */
int num_seg = 1;   /* segment counter */

float t[ROW][COL];/* transformed matrix */

main()
{
    Initialize_Graphics_Mode();

    /* read and make segment */
    Read_Make_Seg();

    /* display original image segment no. 1-4 */
    Display_Seg(1);
    Out_Text_XY(10,110,"segment no. 1");
    Display_Seg(2);
    Out_Text_XY(200,50,"segment no. 2");
    Display_Seg(3);
    Out_Text_XY(50,200,"segment no. 3");
    Display_Seg(4);
    Out_Text_XY(400,250,"segment no. 4");
    getchar();
```

```
Clear_Viewport();

Display_Seg(1);        /* display original image segment no. 1 */
Out_Text_XY(10,200,"Original segment no. 1");

/* translate segment no. 1 200 units in x and 200 units in y */
Set_Order(1,1);        /* order 1 means translation */
Set_Transform(1,300,10,1,1,0,0,0,0,0);
Construct_Transform(1);
Display_Seg(1);        /* display translated segment no. 1 */
Out_Text_XY(300,200,"Translated segment no. 1");
getchar();

/* translate back */
Set_Transform(1,-300,-10,1,1,0,0,0,0,0);
Construct_Transform(1);

Clear_Viewport();

Display_Seg(2);    /* display original image segment no. 2 */
Out_Text_XY(200,50,"Original segment no. 2");

/* scale segment no. 2 (double in x), fixed point = (0,0) */
Set_Order(2,2);    /* order 2 means scaling */
Set_Transform(2,0,0,2,2,0,0,0,0,0);
Construct_Transform(2);
Display_Seg(2);    /* display double of segment no. 2 */
Out_Text_XY(400,150,"Scaled segment no. 2");
getchar();

Clear_Viewport();

Display_Seg(3);    /* display original image segment no. 3 */
```

```
    Out_Text_XY(350,250,"Original segment no. 3");

    /* rotate segment no. 3 */
    Set_Order(3,3);    /* order 3 means rotating */
    Set_Transform(3,0,0,1,1,0,0,45,150,250);
    Construct_Transform(3);
    Display_Seg(3);    /* display rotated segment no. 3 */
    Out_Text_XY(200,300,"Rotated segment no. 3");
    getchar();

    Clear_Viewport();

    Display_Seg(1);       /* display original image segment no. 1 */
    Out_Text_XY(10,200,"Original segment no. 1");

    /* translate and rotate 45 degrees */
    Set_Order(1,13);       /* order 13 means translation */
    Set_Transform(1,250,250,1,1,0,0,45,250,250);
    Construct_Transform(1);
    Display_Seg(1);       /* display translated segment no. 1 */
    Out_Text_XY(350,300,"Translated and rotated segment no. 1");

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
```

```
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Move_Rel(int dx,int dy)
{
    moverel(dx,dy);
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
```

```
}

void Draw_Line_Rel(int dx,int dy)
{
    linerel(dx,dy);
}

void Fill_Area(int x,int y)
{
    floodfill(x,y,15);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Clear_Viewport(void)
{
    clearviewport();
}

void Combine_Transformation(float t[ROW][COL],float m[ROW][COL])
/* t = combined transformation matrix */
{
    int r, c;
    float temp[ROW][COL];

    for(r=0 ;r<ROW; r++)
        for(c=0 ;c<COL; c++)
            temp[r][c] = t[r][0]*m[0][c]+t[r][1]*m[1][c]
                        +t[r][2]*m[2][c];
```

```
    for(r=0 ;r<ROW; r++)
        for(c=0 ;c<COL; c++)
            t[r][c] = temp[r][c];

}

void Make_Identity(float m[ROW][COL])
{
    int r,c;

    for(r=0; r<ROW; r++)
        for(c=0; c<COL; c++)
            if (r==c) m[r][c] = 1; else m[r][c] = 0;
}

void Transform_Image(int number,struct entry display_file[SIZE],
        float t[ROW][COL])
{
    int i;
    float temp;

    for(i=seg_table[number].seg_start;
        i<(seg_table[number].seg_length+seg_table[number].seg_start);
        i++)

      if((display_file[i].cmd == 1) || (display_file[i].cmd == 3) ||
         (display_file[i].cmd == 5))
      {
         temp = display_file[i].operand1*t[0][0]
                +display_file[i].operand2*t[1][0]+t[2][0];
         display_file[i].operand2
                = display_file[i].operand1*t[0][1]
                 +display_file[i].operand2*t[1][1]+t[2][1];
         display_file[i].operand1 = temp;
```

```
        }
}

void Translate(int tx,int ty,float t[ROW][COL])
{
    float matrix[ROW][COL];

    Make_Identity(matrix);
    matrix[2][0] = tx; matrix[2][1] = ty;
    Combine_Transformation(t,matrix);
}

void Scale(float sx,float sy,int xf,int yf,float t[ROW][COL])
/* (xf,yf) = fixed point */
{
    float matrix[ROW][COL];

    Make_Identity(matrix);
    matrix[0][0] = sx; matrix[1][1] = sy;
    matrix[2][0] = (1-sx)*xf;
    matrix[2][1] = (1-sy)*yf;
    Combine_Transformation(t,matrix);
}

void Rotate(float degree,int xr,int yr,float t[ROW][COL])
/* (xr,yr) = pivot point */
{
    float matrix[ROW][COL];
    float cdg,sdg,radian;

    Make_Identity(matrix);
    radian = degree*PI/180;
```

```
    cdg = cos(radian); sdg = sin(radian);
    matrix[0][0] = cdg; matrix[0][1] = sdg;
    matrix[1][0] = -sdg; matrix[1][1] = cdg;
    matrix[2][0] = (1-cdg)*xr + yr*sdg;
    matrix[2][1] = (1-cdg)*yr - xr*sdg;
    Combine_Transformation(t,matrix);
}

void Get_Command(void)
{
    int com;            /* commands code */
    int op1,op2;        /* first and second operand */
    FILE    *fin;

    fin = fopen("grcom.dat","r");  /* list of commands file */
    while (!feof(fin)) {
         cmd_index++; /* next available position in display file */
         fscanf(fin,"%d",&com);
         display_file[cmd_index].cmd = com;
         fscanf(fin,"%d",&op1);
         display_file[cmd_index].operand1 = op1;
         fscanf(fin,"%d",&op2);
         display_file[cmd_index].operand2 = op2;
    }
}

void Create_Seg(int number)
{
    if(seg_open==1) {
         printf("Error! : Another segment is open");
         error = 1;
         exit(1); /* exit Create_Seg */
```

```
        }
        if((number<1) || (number>MAX_SEG)) {
              printf("Error! : Invalid segment number");
              error = 1;
              exit(1); /* exit Create_Seg */
        }
        if(seg_table[number].seg_length>0) {
              printf("Error! : Segment already created");
              error = 1;
              exit(1); /* exit Create_Seg */
        }
        /* starting position in display file for segment */
        seg_table[number].seg_start = next_entry;
        /* set default attributes values */
        seg_table[number].visible = 1;     /* display */
        seg_table[number].transorder = 0; /* no transformation */
        seg_table[number].trans_x = 0;     /* no translation */
        seg_table[number].trans_y = 0;
        seg_table[number].scale_x = 1;     /* no scaling */
        seg_table[number].scale_y = 1;
        seg_table[number].fix_x = 0;       /* fixed point at origin */
        seg_table[number].fix_y = 0;
        seg_table[number].rot_angle = 0;  /* no rotation */
        seg_table[number].rot_x = 0;
        seg_table[number].rot_y = 0;       /* pivot point at origin */

        seg_open = 1;  /* segment open ON */
}

void Close_Seg(int number)
{
        if(!seg_open) {
```

```
            printf("Error! : No segment to be closed");
            error = 1;
            exit(1);
    }
    /* set length of a segment (assign the end position) */
    seg_table[number].seg_length
            = next_entry - seg_table[number].seg_start;
    seg_open = 0;   /* segment open OFF */
}

void Make_Seg(int number)
{
    Create_Seg(number);
    next_entry++;
    while (display_file[next_entry].cmd != 0)
            next_entry++;
    Close_Seg(number);
    num_seg++;   /* count segment */
}

void Read_Make_Seg(void)
{
    /* read commands from file into display file */
    Get_Command();

    /* make segment table from display file */
    for(next_entry=0; next_entry<=cmd_index; next_entry++) {
            Make_Seg(num_seg);
    }
    num_seg -= 1;      /* set the correct number of segment */
}
```

```
void Convert_To_Image(int coms,int oper1,int oper2)
{
    switch (coms) {
            case 0 : break;
            case 1 : Move_To(oper1,oper2);
                     break;
            case 2 : Move_Rel(oper1,oper2);
                     break;
            case 3 : Draw_Line_To(oper1,oper2);
                     break;
            case 4 : Draw_Line_Rel(oper1,oper2);
                     break;
            case 5 : Fill_Area(oper1,oper2);
                     break;
    }
}

void Display_Seg(int number)
{
    int i;

    if((number<1) || (number>num_seg)) {
          printf("Error! : Invalid segment number");
          error = 1;
          exit(1); /* exit Create_Seg */
    }
    if(seg_table[number].visible == 1)
    {
    for(i=seg_table[number].seg_start;
        i<(seg_table[number].seg_length+seg_table[number].seg_start);
        i++)
          Convert_To_Image(display_file[i].cmd,
```

```
        display_file[i].operand1,
        display_file[i].operand2);
    }
}

void Set_Transform(int number,int tx,int ty,int sx,int sy,
  int fx,int fy,int rot,int rx,int ry)
{
    if((number<1) || (number>num_seg)) {
        printf("Error! : Invalid segment number");
        error = 1;
        exit(1); /* exit Set_Transform */
    }
    seg_table[number].trans_x = tx;
    seg_table[number].trans_y = ty;
    seg_table[number].scale_x = sx;
    seg_table[number].scale_y = sy;
    seg_table[number].fix_x = fx;
    seg_table[number].fix_y = fy;
    seg_table[number].rot_angle = rot;
    seg_table[number].rot_x = rx;
    seg_table[number].rot_y = ry;
}

void Set_Order(int number,int order)
{
    if((number<1) || (number>num_seg)) {
        printf("Error! : Invalid segment number");
        error = 1;
        exit(1); /* exit Set_Order */
    }
    seg_table[number].transorder = order;
```

```
}

void Set_Visible(int number,int see) /* see = 1 visible,
                                               = 0 invisible */
{
    if((number<1) || (number>num_seg)) {
         printf("Error! : Invalid segment number");
         error = 1;
         exit(1); /* exit Set_Visible */
    }
    seg_table[number].visible = see;
}

void Construct_Transform(int number)
{
    switch(seg_table[number].transorder) {
case 0    : break;
case 1    : Make_Identity(t);
            Translate(seg_table[number].trans_x,
                      seg_table[number].trans_y,t);
            Transform_Image(number,display_file,t);
            break;
case 2    : Make_Identity(t);
            Scale(seg_table[number].scale_x,
                  seg_table[number].scale_y,
                  seg_table[number].fix_x,
                  seg_table[number].fix_y,t);
            Transform_Image(number,display_file,t);
            break;
case 3    : Make_Identity(t);
            Rotate(seg_table[number].rot_angle,
                   seg_table[number].rot_x,
```

```
                seg_table[number].rot_y,t);
                Transform_Image(number,display_file,t);
                break;
case 12 : Make_Identity(t);
          Translate(seg_table[number].trans_x,
                    seg_table[number].trans_y,t);
          Scale(seg_table[number].scale_x,
                seg_table[number].scale_y,
                seg_table[number].fix_x,
                seg_table[number].fix_y,t);
          Transform_Image(number,display_file,t);
          break;
case 13 : Make_Identity(t);
          Translate(seg_table[number].trans_x,
                    seg_table[number].trans_y,t);
          Rotate(seg_table[number].rot_angle,
                 seg_table[number].rot_x,
                 seg_table[number].rot_y,t);
          Transform_Image(number,display_file,t);
          break;
    }
}
```

ต่อไปนี้เป็นสรุปฟังก์ชันที่ใช้งานในโปรแกรมตัวอย่างข้างต้น (สำหรับฟังก์ชันที่เคยใช้มาแล้วในบทต้น ๆ จะไม่อธิบายในที่นี้)

Transform_Image	ใช้สำหรับแปลงค่าพิกัดซึ่งเก็บในโครงสร้างแบบแมทริกส์ไปเป็นพิกัดที่เก็บในโครงสร้างข้อมูลของเซกเมนต์ (ในบทที่ 5 ก็มีฟังก์ชันชื่อนี้เช่นกัน แต่ใช้พารามิเตอร์ต่างกันเล็กน้อยกล่าวคือฟังก์ชันในบทนี้จะต้องกำหนดชื่อของเซกเมนต์ที่จะแปลงด้วย)
Get_Command	ใช้สำหรับอ่านรายการคำสั่งจากไฟล์ grcom.dat เข้ามาเก็บไว้ในไฟล์แสดงภาพ

Create__Seg(num)	ใช้สำหรับกำหนดตำแหน่งเริ่มต้นของเซกเมนต์ชื่อ num ในไฟล์แสดงภาพ และกำหนดค่าเริ่มต้นให้แอตทริบิวต์ต่าง ๆ
Close__Seg(num)	ใช้สำหรับกำหนดความยาวของเซกเมนต์ num
Make__Seg(num)	เป็นฟังก์ชันรวม Create__Seg กับ Close__Seg เข้าด้วยกัน พร้อมทั้งนับจำนวนเซกเมนต์
Read__Make__Seg	เป็นฟังก์ชันรวม Get__Command กับ Make__Seg เข้าด้วยกัน
Convert__To__Image	ใช้สำหรับแปลงรหัสคำสั่งให้เป็นคำสั่งทางกราฟิกส์ที่ใช้งานได้
Display__Seg(num)	ใช้สำหรับแสดงเซกเมนต์ num ให้เป็นภาพบนจอภาพ
Set__Transform	ใช้สำหรับกำหนดพารามิเตอร์ของการแปลงภาพให้กับเซกเมนต์ที่กำหนด
Set__Order	ใช้สำหรับกำหนดแบบของการแปลงภาพให้กับเซกเมนต์ที่กำหนดเช่น กำหนดเป็น 13 หมายถึงให้ย้ายภาพแล้วหมุนภาพเป็นต้น แบบของการแปลงสามารถเพิ่มมากขึ้นได้เช่น เราอาจจะกำหนดเป็น 123 หมายถึงให้ย้ายภาพแล้วย่อขยายภาพแล้วสุดท้ายหมุนภาพ เป็นต้น
Construct__Transform	เป็นฟังก์ชันที่นำผลของการกำหนดแบบของการแปลงจากฟังก์ชัน Set__Order มาจัดการให้ได้ผลตามที่กำหนดไว้
Set__Visible	ใช้กำหนดว่าจะให้แสดงผลภาพหรือไม่

6.6 สรุป

เซกเมนต์เป็นวิธีการหนึ่งที่ช่วยให้การแสดงผลภาพทำได้อิสระมากขึ้น กล่าวคือ ภาพย่อยแต่ละส่วนจะเก็บในรูปของเซกเมนต์ เราสามารถกำหนดแอตทริบิวต์และแสดงภาพย่อยส่วนนั้นได้โดยจัดการกับเซกเมนต์นั้น และการดำเนินการจะมีผลต่อเซกเมนต์นั้นเพียงเซกเมนต์เดียวเท่านั้น

7 การออกแบบโปรแกรมกราฟิกส์

7.1 บทนำ

ในการออกแบบโปรแกรมกราฟิกส์นั้น จุดประสงค์ของการออกแบบก็คือ ระบบที่ออกแบบจะเป็นระบบที่ง่ายต่อการใช้งานมากที่สุด กล่าวคือ โปรแกรมจะต้องช่วยผู้ใช้โดยการมีข้อความหรือสิ่งสื่อความหมายอื่น ๆ เพื่อตอบสนองต่อคำสั่งของผู้ใช้ และควรมีข้อแนะนำถึงขั้นตอนต่อไปด้วยว่าควรจะทำอะไร การออกแบบอุปกรณ์นำเข้าและวิธีสื่อความหมายระหว่างคนกับคอมพิวเตอร์นั้นในปัจจุบันก็ยังมีการพัฒนาศึกษากันอย่างมากมาย ตัวอย่างเช่น อุปกรณ์นำเข้าและรูปแบบการแสดงผลแบบใดจึงจะเหมาะกับผู้ใช้ทั่วไปมากที่สุด เป็นต้น สำหรับในบทนี้เราจะกล่าวถึงเทคนิคต่าง ๆ ที่ยอมรับกันแล้วว่าเป็นเทคนิคที่ทำให้โปรแกรมง่ายต่อการใช้งานมากที่สุด (user-friendly) แต่ก่อนอื่นเราจะกล่าวถึงข้อจำกัดทางฮาร์ดแวร์ที่จะใช้สำหรับวางระบบกราฟิกส์

7.2 หน่วยความจำและความเร็วของฮาร์ดแวร์

สิ่งที่ต้องการในการเขียนโปรแกรมกราฟิกส์ก็คือ ความเร็วในการแสดงผล ซึ่งควรจะมีความเร็วมากที่สุดเท่าที่จะเป็นไปได้และควรจะใช้เนื้อที่ในหน่วยความจำน้อยที่สุด ซึ่งก็มีเทคนิคหลายอย่าง ตัวอย่างเช่น โปรแกรมที่มีขนาดใหญ่ ๆ เราไม่สามารถมีเนื้อที่ในหน่วยความจำเพียงพอที่จะใส่โปรแกรมทั้งหมดลงไปได้ก็จะใช้เนื้อที่ในดิสก์มาใช้แทนหน่วยความจำ ซึ่งเวลาต้องการใช้ส่วนของโปรแกรมที่อยู่ในดิสก์ ก็จะอ่านเข้ามาในหน่วยความจำแล้วใช้งาน เทคนิคนี้ช่วยให้เราสามารถลดขนาดของหน่วยความจำของเครื่องได้ แต่มีผลทำให้โปรแกรมทำงานได้ช้าลง การแสดงผลก็จะช้าลงตามไปด้วย ซึ่งจะไม่เป็นโปรแกรมกราฟิกส์ที่ดี โดยเฉพาะโปรแกรมเกี่ยวกับการแสดงภาพเคลื่อนไหวจะต้องมีหน่วยความจำมากเพียงพอ

การเขียนโปรแกรมกราฟิกส์ ก็มีหลักการเขียนเช่นเดียวกับการเขียนโปรแกรมอื่น ๆ นั่นคือควรมีลักษณะเป็นโปรแกรมโครงสร้างซึ่งต้องการการออกแบบแบบบนลงล่างหรือท็อปดาวน์ (top-down design) ซึ่งช่วยแบ่งโปรแกรมออกเป็นโมดูลย่อย ๆ ได้ แต่การมีโมดูลมาก ๆ ก็จะทำให้โปรแกรมทำงานได้ช้าลง ดังนั้นจึงต้องออกแบบตั้งแต่แรกเลยว่าจะเลือกโมดูลใดเป็นโมดูลหลักและโมดูลใดที่เป็นส่วนเพิ่มเติม และควรมีจำนวนโมดูลเท่าไรจึงจะเหมาะสมที่สุด

7.3 ลักษณะของระบบซึ่งง่ายต่อการใช้งาน

ระบบกราฟิกส์ที่ดี จะต้องง่ายต่อการใช้งานมากที่สุด ผู้ใช้ทั่ว ๆ ไปซึ่งไม่มีประสบการณ์การใช้โปรแกรมนี้เลยจะต้องสามารถเรียนรู้การใช้งานโปรแกรมได้อย่างรวดเร็ว ดังนั้นส่วนประกอบทางฮาร์ดแวร์ของระบบจะต้องเหมาะสม เช่น อุปกรณ์นำเข้าควรเป็น *เมาส์* ซึ่งทำให้เราสามารถจะเคลื่อนที่เคอร์เซอร์ไปที่ตำแหน่งใด ๆ บนจอภาพได้โดยง่าย นอกจากนี้การเลือกภาพที่ต้องการ การเคลื่อนภาพ ก็ทำได้โดยใช้เมาส์ ซึ่งทำให้การติดต่อระหว่างคนกับเครื่องเป็นไปได้อย่างสะดวกใกล้เคียงกับธรรมชาติการทำงานของมนุษย์

นอกจากส่วนประกอบทางฮาร์ดแวร์ที่เหมาะสมแล้ว ส่วนของการแสดงผลที่ใช้สำหรับติดต่อกับผู้ใช้งานจะต้องออกแบบให้ง่ายต่อการทำความเข้าใจมากที่สุดด้วย เช่น ภาษาที่ใช้ตอบโต้กับผู้ใช้ควรเป็นภาษาที่ผู้ใช้งานเข้าใจได้ดี นอกจากนี้กรณีที่ผู้ใช้ทำผิดพลาด เช่น ให้ข้อมูลเข้าโปรแกรมที่ไม่เหมาะสม เช่น โปรแกรมต้องการค่าจำนวนเต็มแต่ผู้ใช้ให้ค่าตัวอักษร โปรแกรมก็ควรจะสามารถแสดงข้อความให้ผู้ใช้ทราบว่าทำอะไรผิดพลาดได้ และอาจจะมีข้อแนะนำว่าควรทำอะไรด้วยก็จะยิ่งดี ตัวอย่างที่กล่าวมาเป็นตัวอย่างของการแสดงผลตอบสนองของโปรแกรมขณะใช้งาน ยังมีส่วนสำคัญอีกส่วนหนึ่งที่โปรแกรมสำเร็จรูปแทบทุกโปรแกรมในปัจจุบันจะต้องมี และโปรแกรมกราฟิกส์ควรจะต้องมีเสมอ ส่วนนี้ก็คือ *เมนู (Menus)* ซึ่งช่วยให้ผู้ใช้ไม่ต้องจำว่าโปรแกรมมีคำสั่งอะไรให้ใช้ได้บ้าง สำหรับผู้ที่ไม่เคยใช้โปรแกรมนี้ก็สามารถจะเรียนรู้การใช้คำสั่งต่าง ๆ ได้ด้วยตัวเอง ต่อไปนี้ก็จะกล่าวถึงเทคนิคที่ทำให้สามารถสร้างโปรแกรมกราฟิกส์ซึ่งมีลักษณะง่ายต่อใช้งาน

7.3.1 การจัดการกับข้อผิดพลาด

ในการป้อนข้อมูลและการสั่งงานของผู้ใช้นั้น ผู้ใช้มีโอกาสที่จะป้อนข้อมูลซึ่งไม่เหมาะสมกับที่โปรแกรมต้องการได้เสมอ เช่น โปรแกรมต้องการค่าตัวเลขแต่ผู้ใช้อาจจะเผลอให้ข้อมูลตัวอักษรไปได้ โปรแกรมที่ดีควรจะสามารถตรวจสอบได้ในทันทีว่าข้อมูลดังกล่าวไม่สามารถนำไป

ใช้ในโปรแกรมได้และตอบสนองด้วยข้อความหรือเสียงเพื่อให้ผู้ใช้ป้อนข้อมูลได้ใหม่อีก จนกว่าผู้ใช้จะให้ข้อมูลที่เหมาะสมกับโปรแกรม และโปรแกรมควรจะไม่หยุดการทำงานเมื่อพบกับข้อผิดพลาดเช่นนี้ เมื่อได้รับข้อมูลถูกต้องแล้วโปรแกรมก็จะทำงานได้ต่อไป

7.3.2 เมนู

เมนูประกอบไปด้วยข้อความและ/หรือรูปภาพสัญลักษณ์ ซึ่งใช้เป็นตัวแทนการทำงานต่าง ๆที่เตรียมไว้แล้วในระบบกราฟิกส์ ถ้าต้องการให้ระบบกราฟิกส์ทำงานใดก็ให้เลือกข้อความหรือรูปภาพสัญลักษณ์ซึ่งแทนการทำงานนั้น เมื่อเลือกแล้วระบบก็จะทำงานตามที่เราต้องการได้ ซึ่งการจัดการเช่นนี้ทำให้ผู้ใช้สามารถสั่งงานระบบกราฟิกส์ได้อย่างง่ายดายและลดความผิดพลาดซึ่งเกิดจากการให้ข้อมูลหรือคำสั่งได้ นอกจากนี้ยังทำให้เกิดความสะดวกในการใช้งานโปรแกรม นั่นคือ ผู้ใช้ไม่จำเป็นต้องจำคำสั่งต่าง ๆ ที่ใช้ในระบบกราฟิกส์เลย เนื่องจากคำสั่งทุกอย่างในระบบกราฟิกส์ปรากฏบนจอภาพแล้ว

การออกแบบเมนูเป็นส่วนที่สำคัญมากของโปรแกรมกราฟิกส์ เนื่องจากถ้าโปรแกรมมีระบบเมนูที่ไม่ดีก็จะไม่มีผู้ใช้มาใช้งานโปรแกรมนี้

เมนูมักจะปรากฏเต็มจอภาพในตอนเริ่มต้นโปรแกรมเท่านั้น หลังจากนั้นจะเป็นเมนูเล็ก ๆ ตรงส่วนใดส่วนหนึ่งของจอภาพ ทั้งนี้ก็เนื่องจากต้องการให้มีเนื้อที่บนจอภาพสำหรับการแสดงผลภาพที่ผู้ใช้สร้างขึ้นนั่นเอง

เมนูซึ่งมีใช้กันปัจจุบันนี้มีด้วยกัน 2 แบบ คือ *แบบดึงลงหรือพูลดาวน์ (pull-down menu)* และ *แบบผุดขึ้นหรือพอปอัป (pop up menu)* ซึ่งมักเป็นข้อความ ส่วนเมนูที่เป็นสัญลักษณ์มักจะปรากฏบนจอภาพตลอดเวลา แต่อยู่ตรงขอบใดขอบหนึ่งของจอภาพ เช่นอยู่ทางขอบขวาของจอภาพ เป็นต้น เมนูนี้อาจจะสามารถสั่งให้แสดงเมนูหรือไม่ก็ได้

ในกรณีที่สั่งให้เมนูปรากฏบนจอภาพอาจจะมีบางส่วนที่ไปทับกับภาพกราฟิกส์ที่ผู้ใช้สร้างขึ้นก็ได้ ระบบกราฟิกส์จะต้องจัดการไม่ให้ภาพส่วนนั้น (ที่ถูกเมนูทับ) เสียหาย กล่าวคือ เมื่อปิดเมนูแล้ว ภาพส่วนนั้นจะต้องคงอยู่ที่ตำแหน่งเดิม

โปรแกรมที่ 7.1 เป็นโปรแกรมสร้างพูลดาวน์เมนูอย่างง่าย ๆ สำหรับเมนูที่เป็นภาพสัญลักษณ์ จะมีตัวอย่างในโปรแกรมที่ 8.2 ของบทต่อไป

โปรแกรมที่ 7.1

```
/* A C program that demonstrates how to create
   a simple pull down menu */

#include <graphics.h>
#include <dos.h>
#include <conio.h>

#define NO_CHOICE 5     /* number of choices in each menu */
#define NO_MENU 5       /* number of menus */

#define UP_ARROW 0x4800       /* scan code (16 bits) for up arrow */
#define DOWN_ARROW 0x5000     /*                          down arrow */
#define LEFT_ARROW 0x4B00     /*                          left arrow */
#define RIGHT_ARROW 0x4D00    /*                         right arrow */
#define RETURN 0x1C0D         /*                          return key */
#define ESCAPE 0x011B         /*                        escape key */

typedef struct heading {  /* data structure of choice heading */
     char *choice;
};

/* data structure of a menu */
typedef struct menu_struct {
     int frame[4];            /* 4 parameters for drawing menu frame */
     int row[NO_CHOICE];      /* x-value of each choice */
     int col;                 /* y-value of each choice */
     struct heading item[NO_CHOICE]; /* index of each item */
     int last_choice;         /* the last choice of each menu */
};

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
```

```
void Draw_Fill_Rectangle(int lt,int tp,int rt,int bt);
void Erase_Fill_Rectangle(int lt,int tp,int rt,int bt);
void Set_Fill_Pattern(int pattern,int color);
void Set_Color(int color);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);

void Menu_Assignment(void);
void Display_Main_Menu(void);
void Display_Menu(int menu_no);
void Select_Menu(int menu_no,int choice_no);
void Unselect_Menu(int menu_no,int choice_no);
int Read_Key(int key);
void Inverse(int menu_no,int choice_no);
void Normal(int menu_no,int choice_no);

void Do_Choice(void);

/* structure of the menus */
struct menu_struct menus[NO_MENU];

int start = 5,
    one_choice_width = 20;

main()
{
    Initialize_Graphics_Mode();

    Menu_Assignment();
    Display_Main_Menu();
    Select_Menu(1,1);
```

```
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getchar();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}
```

```
void Draw_Fill_Rectangle(int lt,int tp,int rt,int bt)
{
    Set_Fill_Pattern(1,15);
    bar(lt,tp,rt,bt);
}
void Erase_Fill_Rectangle(int lt,int tp,int rt,int bt)
{
    Set_Fill_Pattern(1,0);
    bar(lt,tp,rt,bt);
}
void Set_Fill_Pattern(int pattern,int color)
{
    setfillstyle(pattern,color);
}
void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}
void Set_Color(int color)
{
    setcolor(color);
}
int Get_Max_X(void)
{
    return(getmaxx());
}
int Get_Max_Y(void)
{
    return(getmaxy());
}
```

```
void Menu_Assignment(void)
{
    int one_part;

    one_part = (Get_Max_X()/NO_MENU);

    /* assign menu no. 0 menu About */
    menus[0].frame[0] = start;
    menus[0].frame[1] = one_choice_width;
    menus[0].frame[2] = start+one_part;
    menus[0].frame[3] = one_choice_width+one_choice_width*2;

    menus[0].row[0] = 6;
    menus[0].row[1] = 26;
    menus[0].row[2] = 46;
    menus[0].row[3] = 66;
    menus[0].row[4] = 86;

    menus[0].col = start;
    start += one_part;

    menus[0].item[0].choice = " About         ";
    menus[0].item[1].choice = "  Information  ";
    menus[0].item[2].choice = "  Programmers  ";
    menus[0].item[3].choice = "";
    menus[0].item[4].choice = "";

    menus[0].last_choice = 2;

    /* assign menu no. 1 menu File */
    menus[1].frame[0] = start;
    menus[1].frame[1] = one_choice_width;
    menus[1].frame[2] = start+one_part;
    menus[1].frame[3] = one_choice_width+one_choice_width*4;
```

```
menus[1].row[0] = 6;
menus[1].row[1] = 26;
menus[1].row[2] = 46;
menus[1].row[3] = 66;
menus[1].row[4] = 86;

menus[1].col = start;
start += one_part;

menus[1].item[0].choice =  " File          ";
menus[1].item[1].choice =  "  Open         ";
menus[1].item[2].choice =  "  Save         ";
menus[1].item[3].choice =  "  Print        ";
menus[1].item[4].choice =  "  Quit         ";

menus[1].last_choice = 4;

/* assign menu no. 2 menu Edit */
menus[2].frame[0] = start;
menus[2].frame[1] = one_choice_width;
menus[2].frame[2] = start+one_part;
menus[2].frame[3] = one_choice_width+one_choice_width*4;

menus[2].row[0] = 6;
menus[2].row[1] = 26;
menus[2].row[2] = 46;

menus[2].row[3] = 66;
menus[2].row[4] = 86;

menus[2].col = start;
start += one_part;
```

```
menus[2].item[0].choice = " Edit              ";
menus[2].item[1].choice = "  Cut              ";
menus[2].item[2].choice = "  Copy             ";
menus[2].item[3].choice = "  Paste            ";
menus[2].item[4].choice = "  Clear            ";

menus[2].last_choice = 4;

/* assign menu no. 3 menu Transformation */
menus[3].frame[0] = start;
menus[3].frame[1] = one_choice_width;
menus[3].frame[2] = start+one_part;
menus[3].frame[3] = one_choice_width+one_choice_width*3;

menus[3].row[0] = 6;
menus[3].row[1] = 26;
menus[3].row[2] = 46;
menus[3].row[3] = 66;
menus[3].row[4] = 86;

menus[3].col = start;
start += one_part;

menus[3].item[0].choice = " Transformation";
menus[3].item[1].choice = "  Translation   ";
menus[3].item[2].choice = "  Scaling       ";
menus[3].item[3].choice = "  Rotation      ";

menus[3].item[4].choice = "";

menus[3].last_choice = 3;
```

```
    /* assign menu no. 4 menu Help */
    menus[4].frame[0] = start;
    menus[4].frame[1] = one_choice_width;
    menus[4].frame[2] = start+one_part-1;
    menus[4].frame[3] = one_choice_width+one_choice_width*2;

    menus[4].row[0] = 6;
    menus[4].row[1] = 26;
    menus[4].row[2] = 46;
    menus[4].row[3] = 66;
    menus[4].row[4] = 86;

    menus[4].col = start;
    start += one_part;

    menus[4].item[0].choice = " Help          ";
    menus[4].item[1].choice = "  Contents     ";
    menus[4].item[2].choice = "  Index        ";
    menus[4].item[3].choice = "";
    menus[4].item[4].choice = "";

    menus[4].last_choice = 2;
}

void Display_Main_Menu(void)
{
    int i;

    /* draw main menu border */

    Draw_Rectangle(0,0,Get_Max_X(),19);
    /* draw heading */
    for(i=0; i<NO_MENU; i++) {
```

```
        Out_Text_XY(menus[i].col,menus[i].row[0],
            menus[i].item[0].choice);
    }
}

void Display_Menu(int menu_no)
{
    int i;

    Draw_Rectangle(menus[menu_no].frame[0],menus[menu_no].frame[1],
                   menus[menu_no].frame[2],menus[menu_no].frame[3]);
    for(i=1 ;i<NO_CHOICE; i++)
          Out_Text_XY(menus[menu_no].col,menus[menu_no].row[i],
                      menus[menu_no].item[i].choice);
}

void Select_Menu(int menu_no,int choice_no)
{
    int new_menu_no,
        new_choice_no = 0;
    int i;

    /* default menu */
    Display_Menu(menu_no);
    Inverse(menu_no,choice_no);

    for(;;) {
    i = Read_Key(0);  /* read keyboard */
      switch (i) {
          case LEFT_ARROW  : if(menu_no == 0)
                                     new_menu_no = NO_MENU-1;
                             else new_menu_no = menu_no-1;
                             Unselect_Menu(menu_no,choice_no);
                             Select_Menu(new_menu_no,1);
                             break;
```

```
case RIGHT_ARROW : if(menu_no == NO_MENU-1)
                          new_menu_no = 0;
                   else new_menu_no = menu_no+1;
                   Unselect_Menu(menu_no,choice_no);
                   Select_Menu(new_menu_no,1);
                   break;

case UP_ARROW    : if(choice_no == 1)
                          new_choice_no
                          = menus[menu_no].last_choice;
                   else new_choice_no = choice_no-1;
                   Normal(menu_no,choice_no);
                   Inverse(menu_no,new_choice_no);
                   choice_no = new_choice_no;
                   break;

case DOWN_ARROW  : if(choice_no
                          == menus[menu_no].last_choice)
                              new_choice_no = 1;
                   else new_choice_no = choice_no+1;
                   Normal(menu_no,choice_no);
                   Inverse(menu_no,new_choice_no);
                   choice_no = new_choice_no;
                   break;

case RETURN : switch (menu_no) {
                   case 0 : switch (choice_no) {
                   case 1 : Do_Choice();
                            break;
                   case 2 : Do_Choice();
                            break;
                            };
                            break;
```

```
case 1 : switch (choice_no) {
                case 1 : Do_Choice();
                         break;
                case 2 : Do_Choice();
                         break;
                case 3 : Do_Choice();
                         break;
                case 4 : exit(0);

                     };
                     break;
case 2 : switch (choice_no) {
                case 1 : Do_Choice();
                         break;
                case 2 : Do_Choice();
                         break;
                case 3 : Do_Choice();
                         break;
                case 4 : Do_Choice();
                         break;

                     };
                     break;
case 3 : switch (choice_no) {
                case 1 : Do_Choice();
                         break;
                case 2 : Do_Choice();
                         break;
                case 3 : Do_Choice();
                         break;
                     };
                     break;
```

```
                    case 4 : switch (choice_no) {
                                  case 1 : Do_Choice();
                                           break;
                                  case 2 : Do_Choice();
                                           break;
                                        };
                                        break;
                                        }

                    case ESCAPE : exit(0);
                                  break;

      }  /* switch */
   }    /* for */
}

void Inverse(int menu_no,int choice_no)
{
   Draw_Fill_Rectangle(menus[menu_no].col+2,
                       menus[menu_no].row[choice_no],
                       menus[menu_no].col+124,
                       menus[menu_no].row[choice_no]+10);
   Set_Color(0);
   Out_Text_XY(menus[menu_no].col,menus[menu_no].row[choice_no],
                     menus[menu_no].item[choice_no].choice);
   Set_Color(15);
}

void Normal(int menu_no,int choice_no)
{
   Erase_Fill_Rectangle(menus[menu_no].col+2,
                        menus[menu_no].row[choice_no],
                        menus[menu_no].col+124,
                        menus[menu_no].row[choice_no]+10);
```

```
    Out_Text_XY(menus[menu_no].col,menus[menu_no].row[choice_no],
                menus[menu_no].item[choice_no].choice);
}

void Unselect_Menu(int menu_no,int choice_no)
{
    Normal(menu_no,choice_no);
    Set_Color(0);
    Display_Menu(menu_no);
    Set_Color(15);
}

int Read_Key(int key)
{
    return bioskey(key);
}

void Do_Choice(void)
{
    int x,y;

    x = Get_Max_X()/2;
    y = Get_Max_Y()/2;

    Out_Text_XY(x,y,"Do Choice");
    delay(400);
    Set_Color(0);
    Out_Text_XY(x,y,"Do Choice");
    Set_Color(15);
}
```

7.4 สรุป

การออกแบบโปรแกรมกราฟิกส์จะต้องคำนึงถึงทั้งทางด้านฮาร์ดแวร์และทางด้านซอฟต์แวร์ ทางด้านฮาร์ดแวร์ หน่วยความจำและความเร็วเป็นเรื่องที่ต้องพิจารณา ส่วนทางด้านซอฟต์แวร์นั้นการเขียนโปรแกรมให้ง่ายต่อการใช้งานเป็นเรื่องที่ต้องทำ ซึ่งได้แก่การออกแบบเมนู และการจัดการกับข้อผิดพลาด

8

เทคนิคการติดต่อแบบอินเตอร์แอกชั่น

8.1 บทนำ

ในระบบกราฟิกส์แบบอินเตอร์แอกตีฟนั้น ผู้ใช้สามารถควบคุมและแสดงภาพได้โดยใช้อุปกรณ์นำข้อมูลเข้าที่เหมาะสม เช่น เมาส์ ภาพจะไม่เปลี่ยนแปลงจนกว่าผู้ใช้จะให้คำสั่งว่าจะทำอะไร เช่น คำสั่งให้เคลื่อนย้ายภาพไปที่ตำแหน่งต่าง ๆ บนจอภาพ เป็นต้น

สำหรับการแสดงภาพเคลื่อนไหวซึ่งจะกล่าวถึงในบทต่อไปนั้น จะแตกต่างกับการติดต่อแบบอินเตอร์แอกชั่นตรงที่ภาพสามารถเคลื่อนที่ไปได้ แม้ว่าจะไม่ได้รับคำสั่งจากผู้ใช้โดยตรง เช่น ภาพที่ปรากฏในวิดีโอเกมประเภทยานอวกาศต่อสู้กับยานต่างดาว ผู้เล่นเกมสามารถบังคับการเคลื่อนที่ของภาพยานอวกาศของเราเท่านั้น แต่การเคลื่อนที่ของยานต่างดาวซึ่งมาโจมตียานของเรานั้นโปรแกรมจะเป็นตัวกำหนดการเคลื่อนที่ให้เอง

ความเร็วของการแสดงผลภาพ ในระบบกราฟิกส์แบบอินเตอร์แอกตีฟจะเป็นแฟกเตอร์ที่สำคัญมาก ถ้าระบบกราฟิกส์สามารถตอบสนองคำสั่งของผู้ใช้ได้ภายในเวลา $\frac{1}{10}$ วินาที ผู้ใช้จะรู้สึกว่าระบบตอบสนองได้ในทันทีที่ให้คำสั่ง อย่างไรก็ตาม ผู้ใช้ก็อาจจะยอมรับการแสดงผลที่ช้าเล็กน้อยได้ในงานบางอย่าง เช่น เมื่อผู้ใช้กำหนดจุดต่าง ๆ ที่จะให้ลากเส้นแล้วก็อาจจะรอให้เส้นถูกวาดขึ้นมาให้ได้แต่ก็จะรู้สึกไม่พอใจกับการทำงานเช่นนี้เลย ความเร็วในการแสดงผลภาพหรือการตอบสนองของระบบจะขึ้นอยู่กับความซับซ้อนของภาพที่แสดงและความซับซ้อนของคำสั่งที่ให้ เช่น การเคลื่อนย้ายภาพเส้นเดียวจะใช้เวลาน้อยกว่าการเคลื่อนย้ายภาพรถ ซึ่งประกอบด้วยเส้นมากมาย ในบทนี้เราจะกล่าวถึงเทคนิคหลาย ๆ อย่างซึ่งใช้ในระบบกราฟิกส์แบบอินเตอร์แอกตีฟ และมีบางเทคนิคที่นำไปใช้ในการทำภาพเคลื่อนไหวได้ด้วย

ระบบกราฟิกส์แบบอินเตอร์แอกตีฟมักจะให้เราสามารถใช้อุปกรณ์นำข้อมูลเข้าได้หลายชนิด เช่น แป้นพิมพ์, ปากกาแสง, แทร็กบอล (trackball), แพดเดิล, จอยสติ๊ก, หรือเมาส์ อุปกรณ์แต่ละตัวก็เหมาะที่จะใช้กับงานที่แตกต่างกันเช่น ปากกาแสงเหมาะสำหรับการเลือกภาพที่ต้องการ ส่วนจอยสติ๊กก็เหมาะสำหรับย้ายภาพให้ไปในตำแหน่งต่าง ๆ บนจอภาพ เป็นต้น สำหรับปากกาแสง, จอยสติ๊ก, แพดเดิล และแทร็กบอล เป็นอุปกรณ์ที่มีให้ใช้กันในระบบกราฟิกส์สมัยก่อน ส่วนในระบบกราฟิกส์ทั่วไปที่ใช้ไมโครคอมพิวเตอร์มักจะไม่ใช้อุปกรณ์ดังกล่าวกัน จะมีก็เพียงแป้นพิมพ์กับเมาส์เท่านั้น

8.2 อุปกรณ์รับข้อมูลเข้า

เทคนิคที่ใช้ในระบบกราฟิกส์แบบอินเตอร์แอกตีฟ มักจะแบ่งออกเป็น 2 แบบคือ *การกำหนดตำแหน่ง (positioning)* และ *การเลือก (selecting) หรือ การชี้ (pointing)* สำหรับการกำหนดตำแหน่ง คอมพิวเตอร์จะรับข้อมูลตำแหน่งของพิกเซลของจอภาพจากผู้ใช้ ผ่านทางอุปกรณ์นำข้อมูลเข้า ส่วนการเลือก เราจะใช้อุปกรณ์นำข้อมูลเข้านี้เป็นตัวเลือกภาพที่ต้องการ โดยไม่มีการให้ข้อมูลใด ๆ เข้าเลย เทคนิคทั้งสองนี้มักจะใช้ร่วมกันเสมอ เช่น เราจะเลือกภาพที่ต้องการก่อนแล้วกำหนดตำแหน่งใหม่เพื่อนำภาพที่เลือกไปไว้ในที่ที่กำหนด

แป้นพิมพ์เป็นอุปกรณ์ที่รับข้อมูลได้ทั้งตัวเลขและตัวอักษร นอกจากนี้ยังใช้ในการเลือกได้อีกด้วย เช่น การเลือกเมนูโดยใช้ปุ่มลูกศรบนแป้นพิมพ์ สำหรับเมาส์ก็สามารถใช้ได้ทั้งในการกำหนดตำแหน่งและการเลือก การเคลื่อนที่ของเมาส์จะสัมพันธ์กับเคอร์เซอร์ซึ่งเป็นตัวชี้อยู่บนจอภาพ เคอร์เซอร์ก็คือสัญลักษณ์แทนเมาส์นั่นเอง บนตัวเมาส์จะมีปุ่มสำหรับกด อาจจะมีปุ่มเดียวหรือมากกว่าก็ได้ เมื่อกดปุ่ม เมาส์ก็จะส่งค่าตรรกะไปยังโปรแกรมเพื่อบอกโปรแกรมว่ามีการกดปุ่ม ซึ่งอาจจะหมายถึงการกำหนดตำแหน่งหรือการเลือกก็ได้ขึ้นกับว่าขณะนั้นกำลังอยู่ในการทำงานส่วนใด จะเห็นว่าโปรแกรมจะต้องเวียนมาตรวจสถานะของเมาส์ว่ามีการกดปุ่มใดหรือไม่ การเวียนตรวจสอบสถานะเช่นนี้ของโปรแกรม เรียกว่า*โพลลิงก์ (polling)* โปรแกรมจะต้องเวียนตรวจสอบสถานะให้บ่อยเพียงพอที่จะจับการเปลี่ยนแปลงของสถานะของเมาส์ได้ มิฉะนั้นการกดปุ่มของเมาส์ในบางครั้งจะไม่มีผลต่อการทำงานของโปรแกรมเพราะว่าโปรแกรมพลาดการจับสถานะในจุดนั้น ถ้าหากโปรแกรมไม่ใช้วิธีการเวียนตรวจสอบก็อาจจะใช้วิธีเก็บสถานะต่าง ๆ ลงในคิว (queue) ซึ่งจะเรียกว่าเป็นคิวสถานะก็ได้ แล้วโปรแกรมก็จะไปอ่านสถานะจากคิวนี้และทำงานตามที่ได้รับ วิธีนี้จะไม่มีสถานะใดเลยที่ตกหล่นขาดหายไปแบบวิธีโพลลิงก์ แต่วิธีนี้จะทำให้เสียเวลาในการหาสถานะที่ต้องการ

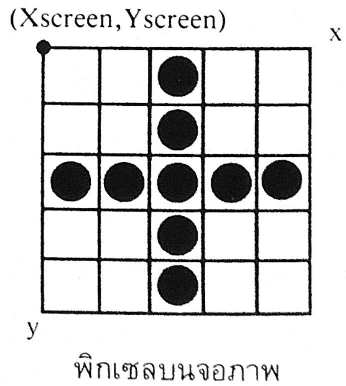
8.3 การเคลื่อนที่ของเคอร์เซอร์

เคอร์เซอร์ (cursor) เป็นภาพสัญลักษณ์ เช่น รูปเครื่องหมายบวกหรือคูณซึ่งปรากฏบนจอภาพใช้เป็นตัวบอกตำแหน่งบนจอภาพ การเคลื่อนที่ของเคอร์เซอร์จะขึ้นกับการบังคับอุปกรณ์นำข้อมูลเข้า เช่น ถ้าเป็นเมาส์ เคอร์เซอร์จะเคลื่อนที่ไปตำแหน่งต่าง ๆ บนจอภาพได้โดยการเคลื่อนที่เมาส์ หรือถ้าใช้แป้นพิมพ์ การกดปุ่มลูกศรซ้ายและขวาจะทำให้เคอร์เซอร์เคลื่อนที่ไปทางซ้ายและขวาตามลำดับ ถ้ากดปุ่มลูกศรขึ้นและลง เคอร์เซอร์ก็จะเคลื่อนที่ขึ้นและลงด้วย

การสร้างรูปเคอร์เซอร์อาจทำได้ง่าย ๆ โดยใช้ฟังก์ชันวาดรูปเคอร์เซอร์ เมื่อต้องการเคลื่อนที่เคอร์เซอร์ก็ให้ลบภาพเคอร์เซอร์ที่ตำแหน่งเดิมแล้ววาดเคอร์เซอร์ที่ตำแหน่งใหม่ในทิศทางที่ต้องการ แต่วิธีการนี้ใช้เวลามากเนื่องจากต้องเรียกใช้ฟังก์ชันนี้ทุกครั้งที่ต้องการวาดรูปเคอร์เซอร์ทำให้การทำงานไม่เป็นอินเตอร์แอกตีฟ ถ้าคอมพิวเตอร์ที่ใช้งานมีความเร็วต่ำ ดังนั้นในระบบกราฟิกส์จะใช้วิธีทำเป็น *ตารางรูปแบบ (shape table)* ซึ่งเป็นตารางรูปแบบของบิตหรือพิกเซลที่เราสามารถอ้างอิงแต่ละพิกเซลและเรียกรูปแบบทั้งหมดมาแสดงผลได้ เรามักจะกำหนดให้ตารางรูปแบบนี้เป็นอาร์เรย์ 2 มิติของข้อมูลชนิดตรรกะ (boolean) ค่าตรรกะจริง (true) จะทำให้พิกเซลเป็นจุดสว่าง ส่วนค่าตรรกะเท็จ (false) จะทำให้พิกเซลไม่สว่าง รูปที่ 8.1 แสดงความสัมพันธ์ระหว่างอาร์เรย์ของเคอร์เซอร์กับพิกเซลบนจอภาพ โปรแกรมที่ 8.1 เป็นโปรแกรมสำหรับสร้างภาพเคอร์เซอร์ โดยใช้ตารางรูปแบบและแสดงการเคลื่อนที่ของเคอร์เซอร์โดยใช้ปุ่มลูกศรบนแป้นพิมพ์เป็นตัวบังคับทิศทาง

F	F	T	F	F
F	F	T	F	F
T	T	T	T	T
F	F	T	F	F
F	F	T	F	F

อาร์เรย์ของเคอร์เซอร์



พิกเซลบนจอภาพ

รูปที่ 8.1 ตารางรูปแบบ

โปรแกรมที่ 8.1

```
/* A C program that demonstrates how to create cursor shape and
   how to move cursor left right up and down */
```

```
#include <graphics.h>

#define SHAPE_TABLE_X 10
#define SHAPE_TABLE_Y 10

#define UP_ARROW 0x4800        /* scan code (16 bits) for up arrow */
#define DOWN_ARROW 0x5000      /*                        down arrow */
#define LEFT_ARROW 0x4B00      /*                        left arrow */
#define RIGHT_ARROW 0x4D00     /*                       right arrow */
#define RETURN 0x1C0D          /*                        return key */
#define ESCAPE 0x011B         /*                       escape key */

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Move_To(int x,int y);
void Out_Text_XY(int x,int y,char text[]);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
int Get_Max_X(void);
int Get_Max_Y(void);

void Draw_Cursor_Arrow(int x,int y,int mode);
/* if mode = 0 means erase cursor
   else means draw cursor */
int Read_Key(int key);
void Move_Cursor_Arrow(void);

/* data structure of cursor shape : an arrow */
int cursor_arrow_shape[SHAPE_TABLE_X][SHAPE_TABLE_Y] = {
        1,1,1,1,1,1,0,0,0,0,
        1,1,1,1,1,0,0,0,0,0,
        1,1,1,1,0,0,0,0,0,0,
        1,1,1,1,1,0,0,0,0,0,
        1,1,0,1,1,1,0,0,0,0,
        1,0,0,0,1,1,1,0,0,0,
```

```
        0,0,0,0,0,1,1,1,0,0,
        0,0,0,0,0,0,1,1,1,0,
        0,0,0,0,0,0,0,1,1,1,
        0,0,0,0,0,0,0,0,1,0
};

main()
{
    Initialize_Graphics_Mode();

    Draw_Rectangle(0,0,Get_Max_X(),Get_Max_Y());
    Move_Cursor_Arrow();

    getchar();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
```

```
}
void Close_Graphics_Mode(void)
{
   closegraph();
}
void Plot_Point(int x,int y,int color)
{
   putpixel(x,y,color);
}
void Move_To(int x,int y)
{
   moveto(x,y);
}
void Out_Text_XY(int x,int y,char text[])
{
   outtextxy(x,y,text);
}
void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
   rectangle(lt,tp,rt,bt);
}
int Get_Max_X(void)
{
   return(getmaxx());
}
int Get_Max_Y(void)
{
   return(getmaxy());
}
```

```
void Draw_Cursor_Arrow(int x,int y,int mode)
{
    int i,j;

    Move_To(x,y);
    for(i=0; i<SHAPE_TABLE_X; i++)
          for(j=0; j<SHAPE_TABLE_Y; j++)
               if(cursor_arrow_shape[i][j]==1)
                  if(mode==0)
                       Plot_Point(x+i,y+j,0);
                  else Plot_Point(x+i,y+j,15);
}

int Read_Key(int key)
{
    return bioskey(key);
}

void Move_Cursor_Arrow(void)
{
    int x,y;  /* coordinate of cursor */
    int step = 5; /* no. of step of cursor moving */
    int i;

    x = Get_Max_X()/2;
    y = Get_Max_Y()/2;

    Draw_Cursor_Arrow(x,y,1);

    for(;;) {
    i = Read_Key(0);  /* read keyboard */
      switch (i) {
         case LEFT_ARROW  : Draw_Cursor_Arrow(x,y,0);/* erase cursor */
                            if(x<=5) x = 5;
                            else x -= step;
```

```
                    Draw_Cursor_Arrow(x,y,1);
                    break;

    case RIGHT_ARROW : Draw_Cursor_Arrow(x,y,0);
                    if(x>=(Get_Max_X()-12))
                        x = Get_Max_X()-12;
                    else x += step;
                    Draw_Cursor_Arrow(x,y,1);
                    break;

    case UP_ARROW    : Draw_Cursor_Arrow(x,y,0);
                    if(y<=5) y = 5;
                    else y -= step;
                    Draw_Cursor_Arrow(x,y,1);
                    break;

    case DOWN_ARROW  : Draw_Cursor_Arrow(x,y,0);
                    if(y>=(Get_Max_Y()-12))
                        y = Get_Max_Y()-12;
                    else y += step;
                    Draw_Cursor_Arrow(x,y,1);
                    break;

    case RETURN      : Out_Text_XY(x,y,"Select this area");
                    break;

    case ESCAPE      : exit(0);
                    break;
        } /* switch */
      } /* for */
}
```

จากโปรแกรมข้างต้น ภาพเคอร์เซอร์ได้จากตารางรูปแบบซึ่งออกแบบให้เป็นรูปลูกศร ส่วนการเคลื่อนที่เคอร์เซอร์หรือลูกศรนั้นจะใช้วิธีตรวจสอบว่ามีการกดปุ่มใดของแป้นพิมพ์ ซึ่งถ้าเป็นปุ่มลูกศร ซ้าย ขวา ลง หรือขึ้น ก็ให้ไปทำการปรับค่าพิกัดของเคอร์เซอร์ตามทิศทางที่ถูกต้องกับความหมายของปุ่ม แล้วทำการวาดรูปลูกศรในตำแหน่งถัดมาที่ปรับแล้ว แต่ก่อนที่จะวาดรูปลูกศรในครั้งใหม่นั้นเราต้องลบรูปลูกศรอันเดิมออกไป ซึ่งใช้วิธีวาดรูปลูกศรทับที่เดิมโดยใช้สีเดียวกับสีพื้นหลังทำให้รูปเคอร์เซอร์หายไป

สมมติว่ามีภาพบนจอภาพแล้ว ถ้าเคอร์เซอร์เคลื่อนไปทับบางส่วนของภาพ ส่วนนั้นของภาพก็จะหายไปด้วย ซึ่งในการทำงานจริงไม่ควรเป็นเช่นนี้ แสดงว่าวิธีการเคลื่อนที่เคอร์เซอร์โดยวิธีการลบแล้วเขียนใหม่นี้ไม่เหมาะสมที่จะนำมาใช้ วิธีการที่เหมาะสมก็คือต้องนำบิตของจอภาพส่วนนั้นมาทำการเอ็กซ์คลูซีฟออร์ (exclusive-OR) กับบิตของเคอร์เซอร์ เมื่อเคอร์เซอร์เลื่อนผ่านเข้าไปทับส่วนของภาพก็จะทำให้ตรงส่วนนั้นผิดเพี้ยนไป แต่เมื่อเคอร์เซอร์เลื่อนผ่านออกไปแล้ว ภาพก็จะกลับมาเป็นดังเดิมได้ นอกจากนี้การลบเคอร์เซอร์ในตำแหน่งเดิมแล้วเลื่อนไปวาดเคอร์เซอร์ที่ตำแหน่งใหม่เพื่อทำให้เกิดภาพการเคลื่อนที่ของเคอร์เซอร์ก็สามารถใช้เทคนิคนี้ได้ กล่าวคือ ถ้าเราทำการเอ็กซ์คลูซีฟออร์ภาพเคอร์เซอร์เข้ากับภาพเคอร์เซอร์ที่ปรากฏอยู่เดิมบนจอภาพ ก็จะทำให้ภาพเคอร์เซอร์หายไปได้เช่นกัน เทคนิคนี้จะมีการใช้ในโปรแกรมที่ 8.2

8.4 เทคนิคการสร้างภาพแบบอินเตอร์แอกตีฟ

ในตอนนี้เราจะกล่าวถึงเทคนิคต่าง ๆ หลายเทคนิคในการสร้างภาพแบบอินเตอร์แอกตีฟ ซึ่งเทคนิคเหล่านี้สามารถจะนำไปใช้ประกอบในการสร้างระบบกราฟิกส์ได้

8.4.1 การวาดเส้นตรง

ในการวาดเส้นตรงนั้นเราเริ่มโดยกำหนดจุดตั้งต้นของเส้นนั้นก่อน แล้วเลื่อนเคอร์เซอร์ไปยังจุดปลายของเส้นแล้วกำหนดจุด โปรแกรมจะทำการลากเส้นเชื่อมจุดทั้งสองนั้นให้เอง วิธีนี้เราจะเห็นเส้นก็ต่อเมื่อเราได้กำหนดตำแหน่งของจุดปลายทั้งสองแล้ว

สำหรับการวาดเส้นตรงในแนวนอนหรือแนวดิ่ง เราจะพบว่าเส้นที่เราวาดโดยการกำหนดจุดปลายสองจุดนั้นมักไม่เป็นเส้นแนวนอนหรือแนวดิ่งที่สมบูรณ์ เนื่องจากเป็นการยากที่จะกำหนดจุดปลายอีกจุดหนึ่งให้อยู่ในแนวเดียวกันกับจุดเริ่มต้นได้ วิธีแก้ไขก็คือ ให้ผู้ใช้กำหนดเพียงตำแหน่งเริ่มต้น ส่วนอีกจุดหนึ่งนั้นจะถูกกำหนดโดยการเลื่อนเคอร์เซอร์ของผู้ใช้เอง เช่น เมื่อกำหนดจุดเริ่มต้นแล้วเราก็เลื่อนเคอร์เซอร์ไปในแนวนอนหรือแนวดิ่งซึ่งไม่จำเป็นต้องเป็นแนวที่ตรงกับจุดตั้งต้นก็ได้ จุดที่เคอร์เซอร์เลื่อนไปเราจะรู้ค่าของพิกัดนั้นเสมอ โปรแกรมก็จะจำเฉพาะค่าทางแกน x ถ้าเป็นการลากเส้นในแนวนอน ส่วนเส้นในแนวดิ่ง โปรแกรมจะจำเพียงค่าทางแกน y เท่านั้น ในกรณีเป็น

เส้นในแนวนอนเมื่อกำหนดจุดปลายแล้ว โปรแกรมจะลากเส้นจากจุดตั้งต้นไปยังจุดพิกัดซึ่งได้ค่าทางแกน x จากที่จำไว้ในตอนเลื่อนเคอร์เซอร์ ส่วนค่าทางแกน y จะใช้ค่าเดิมของจุดเริ่มต้นนั่นเอง ผลก็คือได้เส้นแนวนอน การสร้างเส้นแนวดิ่งก็ใช้วิธีการนี้เช่นกัน วิธีนี้ก็จะเห็นเส้นปรากฏเมื่อกำหนดจุดปลายทั้งสองแล้ว

การวาดเส้นตรงอีกแบบที่ช่วยให้ผู้ใช้สามารถรู้ขนาดและทิศทางของเส้นได้ทันทีก็คือ เมื่อกำหนดจุดเริ่มต้นแล้วขณะที่เลื่อนเคอร์เซอร์ไปยังจุดปลายอีกจุดหนึ่งนั้นจะปรากฏเส้นจากจุดตั้งต้นไปยังจุดที่เคอร์เซอร์อยู่ให้เห็นตลอดเวลา ถ้ายังไม่กำหนดจุดปลาย เมื่อเราเลื่อนเคอร์เซอร์ไปเส้นก็จะเปลี่ยนขนาดและทิศทางตามไปด้วย จนเมื่อเรากำหนดจุดปลายแล้ว เส้นจึงจะปรากฏอย่างถาวรเชื่อมระหว่างจุดปลายทั้งสอง ซึ่งเมื่อเลื่อนเคอร์เซอร์ไปก็จะไม่มีเส้นตามไปปรากฏ

การวาดเส้นแบบต่อเนื่องมีวิธีดังนี้ เมื่อวาดเส้นเชื่อมจุดเริ่มต้นกับจุดปลายแล้ว เราจะใช้จุดปลายของเส้นนั้นเป็นจุดเริ่มต้นแล้วลากเส้นต่อไปยังจุดปลายอีกจุดที่เรากำหนด และเราสามารถทำเช่นนี้ต่อไปได้เรื่อย ๆ ก็จะได้เส้นตรงแบบต่อเนื่องยาวเท่าที่เราต้องการ

เทคนิคที่ให้ภาพเส้นตรงปรากฏตลอดเวลาที่เราเลื่อนเคอร์เซอร์นั้นควรนำไปใช้กับการวาดภาพอื่น ๆ ด้วย เนื่องจากเป็นวิธีที่ช่วยให้ผู้ใช้ทำงานได้ง่ายขึ้น

8.4.2 การวาดภาพรูปทรงเรขาคณิต

ในการวาดรูปสี่เหลี่ยม เราสามารถทำได้โดยกำหนดจุดเพียง 2 จุด กล่าวคือ จุดที่อยู่ตรงกันข้ามกันในแนวทแยงนั่นเอง เช่น จุดที่มุมบนซ้ายกับจุดที่มุมล่างขวา เป็นต้น เราเพียงแต่กำหนดจุดที่มุมหนึ่ง แล้วเลื่อนเคอร์เซอร์ไปยังจุดตรงข้าม (ซึ่งระหว่างนี้จะให้มีภาพสี่เหลี่ยมปรากฏหรือไม่ก็ได้ แต่ควรจะมี) เมื่อกำหนดจุดตรงข้าม โปรแกรมก็จะวาดรูปสี่เหลี่ยมโดยใช้จุดที่กำหนด

สำหรับรูปวงกลม เราสร้างได้โดยเริ่มจากกำหนดจุดศูนย์กลาง (Xc,Yc) ของวงกลมก่อนแล้วเลื่อนเคอร์เซอร์ไปยังตำแหน่งใด ๆ ก็ได้ที่อยู่ห่างจากจุดศูนย์กลางไปเป็นระยะเท่ากับรัศมีที่เราต้องการ จากนั้นก็กำหนดจุด (X,Y) โปรแกรมจะทำการคำนวณหาค่ารัศมีของวงกลมให้เองจากสมการ

$$\text{รัศมี} = \mathrm{sqrt}((Xc-X)^2 + (Yc-Y)^2)$$

การสร้างวงกลมอาจจะสร้างได้จากการกำหนดจุด 2 จุด โดยให้ระยะห่างระหว่าง 2 จุดนี้เท่ากับระยะเส้นผ่านศูนย์กลางของวงกลม จุดศูนย์กลางของวงกลมหาได้จากพิกัดของจุดที่อยู่ตรงกลางระหว่างจุดทั้งสอง และรัศมีก็คือระยะครึ่งหนึ่งของเส้นผ่านศูนย์กลาง

การกำหนดจุด 3 จุดก็สามารถสร้างวงกลมได้ ถ้าจุดทั้งสามนี้อยู่บนเส้นรอบวงของวงกลม เราสามารถนำค่าของพิกัดทั้งสามนี้ไปหาค่าพิกัดของจุดศูนย์กลางและรัศมีได้ โดยใช้วิธีเชิงเลข (numerical method) แต่วิธีนี้ไม่เหมาะที่จะนำมาใช้ในงานทั่วไป

สำหรับรูปทรงเรขาคณิตอื่น ๆ เช่น วงรี รูปหกเหลี่ยม เราก็สามารถสร้างได้โดยกำหนดข้อมูลเข้าที่เหมาะสม ส่วนการคำนวณหาค่าต่าง ๆ ของรูปก็ใช้วิธีทางเรขาคณิตนั่นเอง โปรแกรมที่ 8.2 เป็นโปรแกรมแสดงการสร้างเส้นตรงโดยเทคนิคแบบต่าง ๆ รวมทั้งการสร้างสี่เหลี่ยมและวงกลมด้วย พร้อมทั้งแสดงตัวอย่างเมนูอย่างง่าย ๆ ซึ่งมีลักษณะเป็นสัญลักษณ์

โปรแกรมที่ 8.2

```
/* A C program that demonstrates how to create lines in various
   techniques, rectangle (square) and circle and also how to build
   a simple symbolic menu */

#include <graphics.h>
#include <alloc.h>
#include <math.h>

#define SHAPE_TABLE_X 9       /* cursor shape table dimension */
#define SHAPE_TABLE_Y 9

#define COMMAND_TABLE_X 20    /* shape table dimension of the */
#define COMMAND_TABLE_Y 20    /* commands' symbol in the menu */

#define SPACE 3  /* space between each box of choices */

#define MENU_LT 4          /* frame of menu area */
#define MENU_RT 30
#define MENU_TP 4
#define MENU_BT 150
#define DRAW_AREA_LT 35  /* left side of the frame of drawing area */
#define DRAW_AREA_TP 0   /* right side is the max. of x and y */

#define UP_ARROW 0x4800       /* scan code (16 bits) for up arrow */
#define DOWN_ARROW 0x5000     /*                       down arrow */
```

```
#define LEFT_ARROW 0x4B00      /*                    left arrow */
#define RIGHT_ARROW 0x4D00     /*                   right arrow */
#define RETURN 0x1C0D          /*                    return key */
#define ESCAPE 0x011B          /*                   escape key */

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Draw_Line(int x1,int y1,int x2,int y2);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Out_Text_XY(int x,int y,char text[]);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
void Draw_Circle(int xc,int yc,int radius);
int Get_Max_X(void);
int Get_Max_Y(void);
void Set_Viewport(int lt,int tp,int rt,int bt);
void Clear_Viewport(void);

/* size of the image in display area */
unsigned int Image_Size(int lt,int tp,int rt,int bt);

/* save bit image from screen to memory */
void Get_Image(int lt,int tp,int rt,int bt,void *bitmap);

/* put bit image in memory to screen in various operations
   between bit image and bit screen :
        op = 0 or COPY_PUT --> Copy
        op = 1 or XOR_PUT  --> Exclusive or
        op = 2 or OR_PUT   --> Inclusive or
        op = 3 or AND_PUT  --> And
        op = 4 or NOT_PUT  --> Copy the inverse of the source */
void Put_Image(int lt,int tp,void *bitmap,int op);
```

```
void Save_Cursor(void);
void Save_Commands(void);
void Draw_Menu(void);
void Draw_Image(int x,int y,void *shape,int operator);
int Read_Key(int key);

/* arrow cursor for commands selecting mode */
void Move_Cursor_Arrow(void);
/* plus cursor for drawing mode */
void Move_Cursor_Plus(int comm_no);

void Set_Menu_Inverse(int comm_no); /* set selected choice inverse */
void Set_Menu_Normal(int comm_no);  /* set to normal if unselected */

/* operate drawing commands */
void Do_Command(int x,int y,int comm_no);

/* data structure of selection mode cursor shape : an arrow */
int cursor_arrow_shape[SHAPE_TABLE_X][SHAPE_TABLE_Y] = {
            1,1,1,1,1,0,0,0,0,
            1,1,1,1,0,0,0,0,0,
            1,1,1,1,0,0,0,0,0,
            1,1,1,1,1,0,0,0,0,
            1,0,0,1,1,1,0,0,0,
            0,0,0,0,1,1,1,0,0,
            0,0,0,0,0,1,1,1,0,
            0,0,0,0,0,0,1,1,1,
            0,0,0,0,0,0,0,1,1
};

/* data structure of drawing mode cursor shape : a plus sign */
int cursor_plus_shape[SHAPE_TABLE_X][SHAPE_TABLE_Y] = {
            0,0,0,0,1,0,0,0,0,
            0,0,0,0,1,0,0,0,0,
            0,0,0,0,1,0,0,0,0,
```

```
        0,0,0,0,1,0,0,0,0,
        1,1,1,1,0,1,1,1,1,
        0,0,0,0,1,0,0,0,0,
        0,0,0,0,1,0,0,0,0,
        0,0,0,0,1,0,0,0,0,
        0,0,0,0,1,0,0,0,0
};

void *arrow;          /* bitmap of the arrow cursor */
void *plus;           /* bitmap of the plus sign cursor */
void *comm[6];        /* array of bitmap of graphics commands */
int menu_pos_x[6]; /* position x of each element in the menu */
int menu_pos_y[6]; /* position y of each element in the menu */

main()
{
   Initialize_Graphics_Mode();

   Save_Cursor();    /* save cursor shape in the memory */
   Save_Commands(); /* save the commands symbol in the memory */
   Draw_Menu();
   Move_Cursor_Arrow(); /* drawing commands selection mode */

   Close_Graphics_Mode();
   return(0);
}

void Initialize_Graphics_Mode(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");
   /* check if error */
   errorcode = graphresult();
```

```
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Plot_Point(int x,int y,int color)
{
    putpixel(x,y,color);
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
```

```
{
    outtextxy(x,y,text);
}
void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}
void Draw_Circle(int xc,int yc,int radius)
{
    circle(xc,yc,radius);
}
int Get_Max_X(void)
{
    return(getmaxx());
}
int Get_Max_Y(void)
{
    return(getmaxy());
}
void Set_Viewport(int lt,int tp,int rt,int bt)
{
    setviewport(lt,tp,rt,bt,1);
}
void Clear_Viewport(void)
{
    clearviewport();
}
unsigned int Image_Size(int lt,int tp,int rt,int bt)
{
```

```
    return(imagesize(lt,tp,rt,bt));
}

void Get_Image(int lt,int tp,int rt,int bt,void *bitmap)
{
    getimage(lt,tp,rt,bt,bitmap);
}

void Put_Image(int lt,int tp,void *bitmap,int op)
{
    putimage(lt,tp,bitmap,op);
}

void Save_Cursor(void)
{
    int i,j;
    unsigned int size;

    /* save cursor arrow */
    Move_To(0,0);
    for(i=0; i<SHAPE_TABLE_X; i++)
        for(j=0; j<SHAPE_TABLE_Y; j++)
            if(cursor_arrow_shape[i][j]==1)
                Plot_Point(i,j,15);

    /* calculate the size of the cursor,
       and allocate memory space for it */
    size = Image_Size(0,0,SHAPE_TABLE_X,SHAPE_TABLE_Y);
    arrow = malloc(size);

    /* grab the arrow cursor image */
    Get_Image(0,0,SHAPE_TABLE_X,SHAPE_TABLE_Y,arrow);
    Put_Image(0,0,arrow,XOR_PUT); /* to erase the cursor image */

    /* save cursor plus sign */
    Move_To(0,0);
```

```
    for(i=0; i<SHAPE_TABLE_X; i++)
        for(j=0; j<SHAPE_TABLE_Y; j++)
            if(cursor_plus_shape[i][j]==1)
                Plot_Point(i,j,15);

    /* calculate the size of the cursor,
       and allocate memory space for it */
    size = Image_Size(0,0,SHAPE_TABLE_X,SHAPE_TABLE_Y);
    plus = malloc(size);

    /* grab the arrow cursor image */
    Get_Image(0,0,SHAPE_TABLE_X,SHAPE_TABLE_Y,plus);
    Put_Image(0,0,plus,XOR_PUT);
}

void Save_Commands(void)
{
    unsigned int size;

    /* save command no. 0 : line between 2 end points */
    Draw_Rectangle(0,0,20,20);
    Plot_Point(5,15,15);
    Draw_Line(13,5,17,5);
    Draw_Line(15,3,15,7);

    /* calculate the size of the image,
       and allocate memory space for it */
    size = Image_Size(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y);
    comm[0] = malloc(size);
    /* grab the image */
    Get_Image(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y,comm[0]);
    Draw_Image(0,0,comm[0],1); /* to erase the image */

    /* save command no. 1 : horizontal line between 2 end points */
    Draw_Rectangle(0,0,20,20);
```

```
Draw_Line(5,10,15,10);
/* calculate the size of the image,
   and allocate memory space for it */
size = Image_Size(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y);
comm[1] = malloc(size);
/* grab the image */
Get_Image(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y,comm[1]);
Draw_Image(0,0,comm[1],1);

/* save command no. 2 : vertical line between 2 end points */
Draw_Rectangle(0,0,20,20);
Draw_Line(10,5,10,15);
/* calculate the size of the image,
   and allocate memory space for it */
size = Image_Size(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y);
comm[2] = malloc(size);
/* grab the image */
Get_Image(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y,comm[2]);
Draw_Image(0,0,comm[2],1);

/* save command no. 3 : clear viewport */
Draw_Rectangle(0,0,20,20);
Out_Text_XY(7,7,"C");

/* calculate the size of the image,
   and allocate memory space for it */
size = Image_Size(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y);
comm[3] = malloc(size);
/* grab the image */
Get_Image(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y,comm[3]);
Draw_Image(0,0,comm[3],1);

/* save command no. 4 : rectangle & square drawing */
Draw_Rectangle(0,0,20,20);
```

```
    Draw_Rectangle(5,5,15,15);
    /* calculate the size of the image,
       and allocate memory space for it */
    size = Image_Size(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y);
    comm[4] = malloc(size);
    /* grab the image */
    Get_Image(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y,comm[4]);
    Draw_Image(0,0,comm[4],1);

    /* save command no. 5 : circle drawing */
    Draw_Rectangle(0,0,20,20);
    Plot_Point(10,10,15);
    Draw_Circle(10,10,5);
    /* calculate the size of the image,
       and allocate memory space for it */
    size = Image_Size(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y);
    comm[5] = malloc(size);
    /* grab the image */
    Get_Image(0,0,COMMAND_TABLE_X,COMMAND_TABLE_Y,comm[5]);
    Draw_Image(0,0,comm[5],1);
}

void Draw_Menu(void)
{
    int i;

    /* draw menu area */
    Draw_Rectangle(MENU_LT,MENU_TP,MENU_RT,MENU_BT);
    for(i=0; i<6; i++) {
       menu_pos_x[i] = MENU_LT+SPACE;
       menu_pos_y[i] = MENU_TP+(i*COMMAND_TABLE_Y+SPACE)+(i*SPACE);
       Draw_Image(menu_pos_x[i],menu_pos_y[i],comm[i],0);
    }

    /* draw drawing area */
```

```
    Draw_Rectangle(DRAW_AREA_LT,DRAW_AREA_TP,Get_Max_X(),Get_Max_Y());
}

void Draw_Image(int x,int y,void *shape,int operator)
{
    Put_Image(x,y,shape,operator);
}

int Read_Key(int key)  /* read keyboard */
{
    return bioskey(key);
}

void Move_Cursor_Arrow(void)
{
    int x,y;        /* coordinate of cursor */
    int step = 5; /* no. of step of cursor moving */
    int i;

    x = DRAW_AREA_LT+5; /* set the beginning position */
    y = DRAW_AREA_TP+5; /* of the arrow cursor        */

    Set_Viewport(0,0,Get_Max_X(),Get_Max_Y());
    Draw_Image(x,y,arrow,1);  /* draw arrow cursor */

    for(;;) {   /* forever loop */
    i = Read_Key(0);  /* read keyboard */
      switch (i) {
case LEFT_ARROW  : Draw_Image(x,y,arrow,1);
                   if(x<=5) x = 5;
                   else x -= step;
                   Draw_Image(x,y,arrow,1);
                   break;
case RIGHT_ARROW : Draw_Image(x,y,arrow,1);
                   if(x>=(Get_Max_X()-12))
```

```
                    x = Get_Max_X()-12;
                else x += step;
                Draw_Image(x,y,arrow,1);
                break;
case UP_ARROW   : Draw_Image(x,y,arrow,1);
                if(y<=5) y = 5;
                else y -= step;
                Draw_Image(x,y,arrow,1);
                break;
case DOWN_ARROW : Draw_Image(x,y,arrow,1);
                if(y>=(Get_Max_Y()-12))
                y = Get_Max_Y()-12;
                else y += step;
                Draw_Image(x,y,arrow,1);
                break;
case RETURN     : /* check arrow cursor is in choice area? */
                if((x>=7)&&(x<=27)) {
                /* select choice no. 0 */
  if ((y>=7)&&(y<=27)) {
                Draw_Image(x,y,arrow,1);
                Set_Menu_Inverse(0);
                /* goto drawing mode */
                Move_Cursor_Plus(0);
   }
   else /* select choice no. 1 */
   if ((y>=30)&&(y<=50)) {
        Draw_Image(x,y,arrow,1);
        Set_Menu_Inverse(1);
        Move_Cursor_Plus(1);
   }
   else /* select choice no. 2 */
   if ((y>=53)&&(y<=73)) {
```

```
          Draw_Image(x,y,arrow,1);
          Set_Menu_Inverse(1);
          Move_Cursor_Plus(1);
      }
      else /* select choice no. 2 */
      if ((y>=53)&&(y<=73)) {
            Draw_Image(x,y,arrow,1);
            Set_Menu_Inverse(2);
            Move_Cursor_Plus(2);
      }
      else /* select choice no. 3 */
      if ((y>=76)&&(y<=96)) {
            Draw_Image(x,y,arrow,1);
            Set_Menu_Inverse(3);
            Move_Cursor_Plus(3);
      }
      else /* select choice no. 4 */
      if ((y>=99)&&(y<=119)) {
            Draw_Image(x,y,arrow,1);
            Set_Menu_Inverse(4);
            Move_Cursor_Plus(4);
      }
      else break;
   }
   else break;
case ESCAPE : exit(0); /* exit the program */
              break;
     } /* switch */
   } /* for */
}
```

```
void Set_Menu_Inverse(int comm_no)
{
   Draw_Image(menu_pos_x[comm_no],menu_pos_y[comm_no],
              comm[comm_no],4);

}
void Move_Cursor_Plus(int comm_no)
{
   int x,y;        /* coordinate of cursor */
   int step = 5; /* no. of step of cursor moving */
   int i;

   x = Get_Max_X()/2;  /* set the biginning position */
   y = Get_Max_Y()/2;  /* of the plus cursor */

   /* set drawing area */
   Set_Viewport(DRAW_AREA_LT+2,DRAW_AREA_TP+2,
                Get_Max_X()-2,Get_Max_Y()-2);

   /* if clear viewport command, no need to draw plus cursor */
   if (comm_no!=3) Draw_Image(x,y,plus,1);

   for(;;) {
    i = Read_Key(0);   /* read keyboard */
     switch (i) {
case LEFT_ARROW   : Draw_Image(x,y,plus,1);
                    if(x<=5) x = 5;
                    else x -= step;
                    Draw_Image(x,y,plus,1);
                    break;
case RIGHT_ARROW  : Draw_Image(x,y,plus,1);
                    if(x>=(Get_Max_X()-12))
                    x = Get_Max_X()-12;
                    else x += step;
```

```
                    Draw_Image(x,y,plus,1);
                    break;
case UP_ARROW     : Draw_Image(x,y,plus,1);
                    if(y<=5) y = 5;
                    else y -= step;
                    Draw_Image(x,y,plus,1);
                    break;
case DOWN_ARROW   : Draw_Image(x,y,plus,1);
                    if(y>=(Get_Max_Y()-12))
                    y = Get_Max_Y()-12;
                    else y += step;
                    Draw_Image(x,y,plus,1);
                    break;
case RETURN : switch (comm_no) {
                   case 0 : Move_To(x+4,y+4);
                            Plot_Point(x+4,y+4,15);
                            Do_Command(x,y,comm_no);
                            break;
                   case 1 : Move_To(x+4,y+4);
                            Plot_Point(x+4,y+4,15);
                            Do_Command(x,y,comm_no);
                            break;
                   case 2 : Move_To(x+4,y+4);
                            Plot_Point(x+4,y+4,15);
                            Do_Command(x,y,comm_no);
                            break;
                   case 3 : Clear_Viewport();
                            Set_Menu_Normal(3);
                            Move_Cursor_Arrow();
                            break;
                   case 4 : Move_To(x+4,y+4);
                            Plot_Point(x+4,y+4,15);
```

```
                            Do_Command(x,y,comm_no);
                            break;
                   case 5 : Move_To(x+4,y+4);
                            Do_Command(x,y,comm_no);
                            break;

   }
   break;
case ESCAPE : Move_Cursor_Arrow();/* goto command mode */
              break;
     } /* switch */
   } /* for */
}

void Set_Menu_Normal(int comm_no)
{
   Set_Viewport(0,0,Get_Max_X(),Get_Max_Y());
   Draw_Image(menu_pos_x[comm_no],menu_pos_y[comm_no],
              comm[comm_no],3);
   Draw_Image(menu_pos_x[comm_no],menu_pos_y[comm_no],
              comm[comm_no],1);
   Move_Cursor_Arrow();
}

void Do_Command(int x,int y,int comm_no)
{
   int i,step = 5;
   int old_x,old_y;
   int r; /* radius of a circle */
   int dx,dy;
   old_x = x;
   old_y = y;

   for(;;) {
   i = Read_Key(0);   /* read keyboard */
```

```
        switch (i) {
  case LEFT_ARROW  : Draw_Image(x,y,plus,1);
                     if(x<=5) x = 5;
                     else x -= step;
                     Draw_Image(x,y,plus,1);
                     break;
  case RIGHT_ARROW : Draw_Image(x,y,plus,1);
                     if(x>=(Get_Max_X()-20))
                     x = Get_Max_X()-20;
                     else x += step;
                     Draw_Image(x,y,plus,1);
                     break;
  case UP_ARROW    : Draw_Image(x,y,plus,1);
                     if(y<=5) y = 5;
                     else y -= step;
                     Draw_Image(x,y,plus,1);
                     break;
  case DOWN_ARROW  : Draw_Image(x,y,plus,1);
                     if(y>=(Get_Max_Y()-12))
                     y = Get_Max_Y()-12;
                     else y += step;
                     Draw_Image(x,y,plus,1);
                     break;
  case RETURN : switch (comm_no) {
        case 0 : Draw_Image(x,y,plus,1);
                 Draw_Line_To(x+4,y+4);
                 Set_Menu_Normal(0);
                 break;
        case 1 : Draw_Image(x,y,plus,1);
                 Draw_Line_To(x+4,old_y+4);
                 Set_Menu_Normal(1);
                 break;
```

```
    case 2 : Draw_Image(x,y,plus,1);
             Draw_Line_To(old_x+4,y+4);
             Set_Menu_Normal(2);
             break;
    case 4 : Draw_Image(x,y,plus,1);
             Draw_Rectangle(old_x+4,old_y+4,
             x+4,y+4);
             Set_Menu_Normal(4);
             break;
    case 5 : Draw_Image(x,y,plus,1);
             dx = (x-old_x)*(x-old_x);
             dy = (y-old_y)*(y-old_y);
             r = sqrt(dx+dy);
             Draw_Circle(old_x+4,old_y+4,r);
             Set_Menu_Normal(5);
             break;
  }
  break;
case ESCAPE : Move_Cursor_Plus(comm_no);
             break;
    } /* switch */
   }
}
```

โปรแกรมที่ 8.2 เป็นการสร้างเมนูสัญลักษณ์อย่างง่าย ๆ สัญลักษณ์หนึ่ง ๆ หมายถึงคำสั่งทางกราฟิกส์ 1 คำสั่ง เช่น คำสั่งวาดวงกลม วาดสี่เหลี่ยม ก็จะใช้สัญลักษณ์รูปวงกลมและสัญลักษณ์เป็นรูปสี่เหลี่ยมตามลำดับ การทำงานของโปรแกรมเป็นดังนี้

โปรแกรมจะตรวจว่ามีการกดปุ่มลูกศรหรือไม่ ถ้ามีก็ให้เคลื่อนเคอร์เซอร์รูปลูกศรไปในทิศทางต่าง ๆ แต่ถ้ามีการกดปุ่ม ESC ก็ให้จบการทำงาน ถ้ามีการกดปุ่ม ENTER หรือ RETURN ในพื้นที่ซึ่งเป็นสัญลักษณ์ในเมนู ก็จะตรวจสอบว่าเป็นพื้นที่ของคำสั่งใด จากนั้นจะไปทำงานในอีกฟังก์ชันหนึ่งซึ่งใช้เคอร์เซอร์เป็นรูปเครื่องหมายบวก ในฟังก์ชันนี้ก็จะมีการตรวจ

สอบการกดปุ่มแป้นพิมพ์เช่นกัน ซึ่งทำให้สามารถเลื่อนเคอร์เซอร์ได้ และมีการตรวจปุ่ม ESC เพื่อออกจากฟังก์ชันนี้กลับไปทำงานในฟังก์ชันที่ใช้ลูกศรเป็นเคอร์เซอร์ แต่ถ้ากดปุ่ม ENTER ก็จะออกไปยังฟังก์ชันใหม่ซึ่งเป็นการทำคำสั่งกราฟิกส์ ซึ่งในโปรแกรมนี้จะเป็นคำสั่งซึ่งต้องกำหนดจุด 2 ครั้ง เช่น การลากเส้นตรง การวาดรูปสี่เหลี่ยม การวาดรูปวงกลม เป็นต้น

8.4.3 การสร้างตารางรูปแบบ

ในหัวข้อ 8.3 เราได้ทราบมาแล้วว่า การแสดงภาพโดยใช้ตารางรูปแบบจะทำได้รวดเร็วกว่าการแสดงภาพที่วาดด้วยวิธีเวกเตอร์ แม้ว่าจะไม่สามารถย่อขยายภาพและหมุนภาพนั้นได้แบบวิธีเวกเตอร์ก็ตาม การแสดงภาพโดยใช้ตารางรูปแบบก็เหมาะกับการแสดงภาพในบางกรณี เช่น กรณีแสดงภาพสำหรับใช้เป็นเมนู หรือภาพที่ไม่ต้องการหมุนหรือการเปลี่ยนขนาด เป็นต้น

การสร้างภาพที่จะเก็บในตารางรูปแบบนั้นเราอาจจะใช้กระดาษกราฟมาช่วยในการออกแบบรูปได้ โดยใช้ช่องสี่เหลี่ยมเล็ก ๆ ของกระดาษกราฟแทน 1 พิกเซล ถ้าระบายช่องนี้ก็หมายถึงเป็นพิกเซลที่ต้องการให้สว่าง ส่วนช่องที่ไม่ระบายก็จะเป็นพิกเซลที่ไม่ต้องการให้สว่าง เนื่องจากการใช้กระดาษกราฟช่วยในการออกแบบนี้ ไม่ค่อยสะดวกนักในการทำงาน จึงมีโปรแกรมที่ใช้สำหรับออกแบบรูปซึ่งอาจเรียกว่า โปรแกรมสร้างสัญลักษณ์รูปแบบก็ได้ โปรแกรมนี้จะช่วยให้ผู้ใช้สามารถออกแบบรูปสัญลักษณ์หรือตัวอักษรแบบแรสเตอร์ได้โดยตรง และสามารถเก็บรูปแบบที่ออกแบบเสร็จแล้วลงในตารางรูปแบบได้เลย

โปรแกรมนี้จะสร้างตารางขนาดหนึ่งขึ้นมาเหมือนกับช่องในกระดาษกราฟแสดงบนจอภาพและมีเคอร์เซอร์ซึ่งเราสามารถเลื่อนได้ เมื่อเราเลื่อนเคอร์เซอร์ไปในช่องต่าง ๆ ในตารางจะเปรียบเสมือนเรากำลังเลือกว่าจะระบายช่องใดบ้างในช่องตารางของกราฟนั่นเอง เมื่อเลือกได้แล้ว (ซึ่งอาจจะเป็นการกดอักษรอะไรก็ได้ตามที่โปรแกรมกำหนดเพื่อแสดงว่าเลือก) โปรแกรมจะระบายสีทึบให้กับช่องนั้น หมายถึงต้องการให้พิกเซลที่ตำแหน่งดังกล่าวสว่าง จากนั้นเมื่อออกแบบเสร็จทั้งรูปแล้วก็จะสามารถเก็บรูปแบบนั้นลงในตารางรูปแบบได้ทันที

โปรแกรมที่ 8.3 เป็นโปรแกรมสำหรับสร้างตารางรูปแบบ

```
/* A C program that demonstrates how to create shape table tool */
#include <graphics.h>
#include <alloc.h>
#define CURSOR_SHAPE_X 10      /* cursor shape table dimension */
#define CURSOR_SHAPE_Y 10
```

```
#define SHAPE_TABLE_X 8         /* image shape table dimension */
#define SHAPE_TABLE_Y 20

#define UP_ARROW 0x4800         /* scan code (16 bits) for up arrow */
#define DOWN_ARROW 0x5000       /*                          down arrow */
#define LEFT_ARROW 0x4B00       /*                          left arrow */
#define RIGHT_ARROW 0x4D00      /*                         right arrow */
#define RETURN 0x1C0D           /*                          return key */
#define ESCAPE 0x011B           /*                        escape key */

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Draw_Line(int x1,int y1,int x2,int y2);
void Move_To(int x,int y);
void Out_Text_XY(int x,int y,char text[]);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
int Get_Max_X(void);
int Get_Max_Y(void);
void Set_Viewport(int lt,int tp,int rt,int bt);
void Clear_Viewport(void);

/* size of the image in display area */
unsigned int Image_Size(int lt,int tp,int rt,int bt);

/* save bit image from screen to memory */
void Get_Image(int lt,int tp,int rt,int bt,void *bitmap);

/* put bit image in memory to screen in various operations
   between bit image and bit screen :
        op = 0 or COPY_PUT --> Copy
        op = 1 or XOR_PUT  --> Exclusive or
        op = 2 or OR_PUT   --> Inclusive or
        op = 3 or AND_PUT  --> And
        op = 4 or NOT_PUT  --> Copy the inverse of the source */
```

```
void Put_Image(int lt,int tp,void *bitmap,int op);
void Save_Shape(void);
void Draw_Design_Table(void);
void Draw_Image(int x,int y,void *shape,int operator);
int Read_Key(int key);

/* move cursor to mark the ON/OFF pixel */
void Move_Cursor(void);

/* data structure of cursor shape : a square */
int cursor_square_shape[CURSOR_SHAPE_X][CURSOR_SHAPE_Y] = {
            0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,
            0,0,0,1,1,1,1,1,0,0,
            0,0,0,1,1,1,1,1,0,0,
            0,0,0,1,1,1,1,1,0,0,
            0,0,0,1,1,1,1,1,0,0,
            0,0,0,1,1,1,1,1,0,0,
            0,0,0,1,1,1,1,1,0,0,
            0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0
};

int marker_shape[CURSOR_SHAPE_X][CURSOR_SHAPE_Y] = {
            0,0,0,0,0,0,0,0,0,0,
            0,0,1,1,1,1,1,1,1,0,
            0,0,1,1,1,1,1,1,1,0,
            0,0,1,1,1,1,1,1,1,0,
            0,0,1,1,1,1,1,1,1,0,
            0,0,1,1,1,1,1,1,1,0,
            0,0,1,1,1,1,1,1,1,0,
            0,0,1,1,1,1,1,1,1,0,
            0,0,1,1,1,1,1,1,1,0,
            0,0,0,0,0,0,0,0,0,0
```

```
0,0,0,0,0,0,0,0,0,0
};
int image_shape[SHAPE_TABLE_X][SHAPE_TABLE_Y];

void *square;           /* bitmap of the square cursor */
void *marker;           /* bitmap of the marker */
void *image;            /* bitmap of the image */

int lt,tp,rt,bt;        /* global variable for drawing desinged table */
int image_x,image_y; /* global variable for drawing screen image */

main()
{
    Initialize_Graphics_Mode();
    Save_Shape(); /* save cursor and marker shape in the memory */
    Clear_Viewport();
    /* draw table for designing image and set positions */
    Draw_Design_Table();
    Move_Cursor();
    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
```

```
            printf("Graphics error : %s\n", grapherrormsg(errorcode));
            printf("Press any key to halt : ");
            getch();
            exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Plot_Point(int x,int y,int color)
{
    putpixel(x,y,color);
}

void Draw_Line(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y2);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
```

```
}
int Get_Max_X(void)
{
    return(getmaxx());
}
int Get_Max_Y(void)
{
    return(getmaxy());
}
void Set_Viewport(int lt,int tp,int rt,int bt)
{
    setviewport(lt,tp,rt,bt,1);
}
void Clear_Viewport(void)
{
    clearviewport();
}
unsigned int Image_Size(int lt,int tp,int rt,int bt)
{
    return(imagesize(lt,tp,rt,bt));
}
void Get_Image(int lt,int tp,int rt,int bt,void *bitmap)
{
    getimage(lt,tp,rt,bt,bitmap);
}
void Put_Image(int lt,int tp,void *bitmap,int op)
{
    putimage(lt,tp,bitmap,op);
}
```

```
void Save_Shape(void)
{
    int i,j;
    unsigned int size;
    /* save cursor square */
    Move_To(0,0);
    for(i=0; i<CURSOR_SHAPE_X; i++)
        for(j=0; j<CURSOR_SHAPE_Y; j++)
            if(cursor_square_shape[i][j]==1)
                Plot_Point(i,j,15);
    /* calculate the size of the cursor,
       and allocate memory space for it */
    size = Image_Size(0,0,CURSOR_SHAPE_X,CURSOR_SHAPE_Y);
    square = malloc(size);

    /* grab the cursor image */
    Get_Image(0,0,CURSOR_SHAPE_X,CURSOR_SHAPE_Y,square);
    Draw_Image(0,0,square,XOR_PUT); /* to erase the cursor image */

    /* save marker shape */
    Move_To(0,0);
    for(i=0; i<CURSOR_SHAPE_X; i++)
        for(j=0; j<CURSOR_SHAPE_Y; j++)
            if(marker_shape[i][j]==1)
                Plot_Point(i,j,15);
    /* calculate the size of the cursor,
       and allocate memory space for it */
    size = Image_Size(0,0,CURSOR_SHAPE_X,CURSOR_SHAPE_Y);
    marker = malloc(size);

    /* grab the image */
    Get_Image(0,0,CURSOR_SHAPE_X,CURSOR_SHAPE_Y,marker);
    Draw_Image(0,0,marker,XOR_PUT); /* to erase the cursor image */
```

```
}
void Draw_Design_Table(void)
{
    int i,j;

    /* calculate top-left of the design table */

    lt = Get_Max_X()/4;
    tp = (Get_Max_Y()/2)-(SHAPE_TABLE_Y*10)/2;
    rt = lt+(SHAPE_TABLE_X*10);
    bt = tp+(SHAPE_TABLE_Y*10);

    /* draw table */
    Draw_Rectangle(lt,tp,rt,bt);
    for(i=lt+10; i<rt; i+=10)
         Draw_Line(i,tp,i,bt);
    for(i=tp+10; i<bt; i+=10)
         Draw_Line(lt,i,rt,i);
    /* initialize image shape table */
    for(i=0; i<SHAPE_TABLE_X; i++)
         for(j=0; j<SHAPE_TABLE_Y; j++)
              image_shape[i][j] = 0;
    /* assign screen image position */
    image_x = Get_Max_X()/4*3;
    image_y = (Get_Max_Y()/2)-(SHAPE_TABLE_Y)/2;

    Out_Text_XY(Get_Max_X()/4-10,Get_Max_Y()/4,"DESIGN TABLE");
    Out_Text_XY(Get_Max_X()/4*3-50,Get_Max_Y()/4,"SCREEN SHAPE");
    Out_Text_XY(Get_Max_X()/4*2,Get_Max_Y()/4*3,"PRESS ESC TO EXIT");
}
void Draw_Image(int x,int y,void *shape,int operator)
{
    Put_Image(x,y,shape,operator);
```

```
}

int Read_Key(int key)  /* read keyboard */
{
    return bioskey(key);
}

void Move_Cursor(void)
{
    int x,y;          /* coordinate of cursor */
    int step = 10;    /* no. of step of cursor moving */
    int i;
    int table_x = 0,
        table_y = 0; /* coordinate of the table */

    /* set the beginning position of the cursor on the shape table */
    x = lt;
    y = tp;

    Draw_Image(x,y,square,1);  /* draw square cursor */

    for(;;) {   /* forever loop */
    i = Read_Key(0);  /* read keyboard */
      switch (i) {
case LEFT_ARROW  : Draw_Image(x,y,square,1);
                   if(x<=lt) x = lt;
                   else {x -= step;table_x -= 1;}
                   Draw_Image(x,y,square,1);
                   break;
case RIGHT_ARROW : Draw_Image(x,y,square,1);
                   if(x>=(rt-step)) x = rt-step;
                   else {x += step;table_x += 1;}
                   Draw_Image(x,y,square,1);
                   break;
```

```
case UP_ARROW     : Draw_Image(x,y,square,1);
                    if(y<=tp) y = tp;
                    else {y -= step;table_y -= 1;}
                    Draw_Image(x,y,square,1);
                    break;
case DOWN_ARROW   : Draw_Image(x,y,square,1);
                    if(y>=(bt-step)) y = bt-step;
                    else {y += step;table_y += 1;}
                    Draw_Image(x,y,square,1);
                    break;
case RETURN : if(image_shape[table_x][table_y]==0) {
                    Draw_Image(x,y,marker,1);
                    image_shape[table_x][table_y] = 1;
                    Plot_Point(image_x+table_x,
                    image_y+table_y,15);
    }
    else {
          Draw_Image(x,y,marker,1);
          image_shape[table_x][table_y] = 0;
          Plot_Point(image_x+table_x,
                     image_y+table_y,0);
    }
    break;
case ESCAPE : exit(0); /* exit the program */
              break;
      } /* switch */
   } /* for */
}
```

โปรแกรมข้างต้นจะแสดงตารางเป็นช่องขนาด 10 × 10 จุด แต่ละช่องจะใช้แทน 1 พิกเซล เมื่อเราเลือกกดปุ่ม ENTER ที่ช่องใดซึ่งเป็นช่องว่าง พิกเซลที่สอดคล้องกับช่องนั้นก็จะสว่างขึ้น

แต่ถ้าช่องนั้นมีการระบายสีซึ่งหมายถึงถูกทำให้เป็นพิกเซลสว่างแล้ว เมื่อกดปุ่ม ENTER ตรงช่องนี้ก็จะทำให้พิกเซลตำแหน่งนี้ไม่สว่าง การกดปุ่ม ESC ก็เพื่อจบการทำงานของโปรแกรม

8.5 การเลือกภาพและการกำหนดตำแหน่งใหม่ให้ภาพ

เมื่อสร้างภาพได้แล้วเราอาจจะต้องการจัดการกับภาพที่ได้มานั้น เช่น เคลื่อนภาพไปยังตำแหน่งใหม่ ลบภาพนั้น หรือทำการแปลงแบบต่าง ๆ เป็นต้น การที่จะทำเช่นนี้ได้สิ่งแรกที่ต้องทำก็คือ เราต้องสามารถเลือกภาพส่วนนั้นได้ ซึ่งหมายถึงชี้ได้ว่าภาพใดเป็นภาพที่เราต้องการจะจัดการ เทคนิคที่ใช้ก็คือเราจะเลื่อนเคอร์เซอร์ไปจนจุดศูนย์กลางของเคอร์เซอร์อยู่ในขอบเขตสี่เหลี่ยมที่ภาพนั้นอยู่ภายในก่อน จากนั้นจะทำการเลือกโดยใช้ฟังก์ชันการเลือกซึ่งจะทำการตรวจสอบว่าตำแหน่งของเคอร์เซอร์ปัจจุบันอยู่ภายในขอบเขตสี่เหลี่ยมหรือไม่ ถ้าอยู่ภายในกรอบฟังก์ชันก็จะส่งค่าเลขประจำตัวของภาพนั้นกลับมาเพื่อบอกว่าภาพเลขที่นั้น ๆ อยู่ในกรอบ เช่น ถ้าส่งค่า 1 กลับมาจะหมายถึงว่าเคอร์เซอร์อยู่ภายในกรอบของภาพเลขที่ 1 แต่ถ้าไม่อยู่ในกรอบก็จะส่งค่าอีกค่าหนึ่ง ซึ่งมักกำหนดเป็น 0 กลับมา หมายถึงว่าเคอร์เซอร์ไม่อยู่ในกรอบของภาพใดเลย

หลังจากที่เราเลือกภาพได้แล้ว ขั้นตอนต่อไปอาจจะเป็นการกำหนดตำแหน่งใหม่ที่จะแสดงบนจอภาพให้กับภาพที่เลือกมา เทคนิคที่ใช้ก็คือภาพที่ถูกเลือกจะเลื่อนตามเคอร์เซอร์ไปจนกว่าจะถึงตำแหน่งใหม่ที่ต้องการ (dragging) จากนั้นจะแสดงภาพที่ตำแหน่งนั้น เมื่อเลื่อนเคอร์เซอร์ต่อไปภาพก็จะไม่ตามเคอร์เซอร์อีกแล้วจนกว่าจะมีการเลือกใหม่

8.6 การวาดภาพแบบใช้พู่กัน

การวาดภาพแบบที่มนุษย์คุ้นเคยที่สุดอย่างหนึ่งก็คือ การใช้พู่กันจุ่มสีแล้ววาดภาพที่ต้องการในระบบกราฟิกส์ที่มีความสามารถสูงและราคาแพงมาก ๆ นั้น สามารถจำลองการวาดแบบใช้พู่กันได้แทบจะสมบูรณ์ ผู้วาดเพียงแต่ใช้อุปกรณ์นำเข้าซึ่งมีรูปร่างคล้ายดินสอแล้วทำการวาดบนพื้นกระดานพิเศษ การวาดของผู้วาดจะไปปรากฏอยู่บนจอภาพ อุปกรณ์นำเข้าสามารถเปลี่ยนเป็นดินสอ พู่กัน แปรง หรืออุปกรณ์พ่นสีก็ได้ ข้อดีของการวาดโดยใช้คอมพิวเตอร์ก็คือ เราสามารถลบบางส่วนของภาพได้โดยง่าย สามารถขยายภาพเพื่อแก้ไขบางจุดได้ เป็นต้น สำหรับระบบกราฟิกส์ราคาถูกเราก็สามารถทำการวาดแบบใช้พู่กันได้บ้าง โดยที่เรากำหนดให้พู่กันของเราเป็นรูปแบบที่ง่าย ๆ เช่น เป็นแปรงแบบเส้น เป็นรูปวงกลมเล็กใหญ่ หรือเป็นจุด ๆ เป็นต้น แบบของภาพที่วาดได้จึงมีจำกัด และโดยทั่วไประบบจะแสดงเป็นเมนูแบบของแปรงให้ผู้ใช้เลือกใช้ได้ตามต้องการ

เทคนิคที่ใช้ในการวาดแบบใช้พู่กันนี้ก็คือ การเคลื่อนที่เคอร์เซอร์ไปโดยไม่ต้องลบภาพที่ได้วาดไปแล้วในตำแหน่งที่ผ่านมา รูปแบบของเคอร์เซอร์ที่ใช้ในตอนนี้ก็คือแบบของแปรงที่ใช้ในการวาดภาพนั่นเอง

โปรแกรมที่ 8.4 เป็นโปรแกรมแสดงตัวอย่างการวาดภาพแบบใช้พู่กัน

```
/* A C program that demonstrates how to do painting */

#include <graphics.h>
#include <alloc.h>

#define SHAPE_TABLE_X 9   /* paint brush shape table dimension */
#define SHAPE_TABLE_Y 9

#define UP_ARROW 0x4800       /* scan code (16 bits) for up arrow */
#define DOWN_ARROW 0x5000     /*                      down arrow */
#define LEFT_ARROW 0x4B00     /*                      left arrow */
#define RIGHT_ARROW 0x4D00    /*                     right arrow */
#define RETURN 0x1C0D         /*                      return key */
#define ESCAPE 0x011B         /*                   escape key */

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Plot_Point(int x,int y,int color);
void Move_To(int x,int y);
void Out_Text_XY(int x,int y,char text[]);
void Draw_Rectangle(int lt,int tp,int rt,int bt);
int Get_Max_X(void);
int Get_Max_Y(void);
void Set_Viewport(int lt,int tp,int rt,int bt);
void Clear_Viewport(void);

/* size of the image in display area */
unsigned int Image_Size(int lt,int tp,int rt,int bt);

/* save bit image from screen to memory */
void Get_Image(int lt,int tp,int rt,int bt,void *bitmap);

/* put bit image in memory to screen in various operations
```

```
   between bit image and bit screen :
           op = 0 or COPY_PUT --> Copy
           op = 1 or XOR_PUT  --> Exclusive or
           op = 2 or OR_PUT   --> Inclusive or
           op = 3 or AND_PUT  --> And
           op = 4 or NOT_PUT  --> Copy the inverse of the source */
void Put_Image(int lt,int tp,void *bitmap,int op);
void Save_Cursor(void);
void Draw_Image(int x,int y,void *shape,int operator);
int Read_Key(int key);

/* move cursor for painting */
void Move_Cursor(void);

/* data structure of cursor shape */
int cursor_brush_shape[SHAPE_TABLE_X][SHAPE_TABLE_Y] = {
          1,1,0,0,0,0,0,0,0,
          1,1,1,0,0,0,0,0,0,
          0,1,1,1,0,0,0,0,0,
          0,0,1,1,1,0,0,0,0,
          0,0,0,1,1,1,0,0,0,
          0,0,0,0,1,1,1,0,0,
          0,0,0,0,0,1,1,1,0,
          0,0,0,0,0,0,1,1,1,
          0,0,0,0,0,0,0,1,1
};

void *brush;        /* bitmap of the paint brush cursor */

main()
{
    Initialize_Graphics_Mode();
    Save_Cursor();   /* save cursor shape in the memory */
    Move_Cursor(); /* painting action */
```

```
    Close_Graphics_Mode();
    return(0);
}
void Initialize_Graphics_Mode(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");
    /* check if error */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Plot_Point(int x,int y,int color)
{
    putpixel(x,y,color);
}

void Move_To(int x,int y)
{
```

```
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}

void Draw_Rectangle(int lt,int tp,int rt,int bt)
{
    rectangle(lt,tp,rt,bt);
}

int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}

void Set_Viewport(int lt,int tp,int rt,int bt)
{
    setviewport(lt,tp,rt,bt,1);
}

void Clear_Viewport(void)
{
    clearviewport();
}

unsigned int Image_Size(int lt,int tp,int rt,int bt)
{
    return(imagesize(lt,tp,rt,bt));
```

```
}
void Get_Image(int lt,int tp,int rt,int bt,void *bitmap)
{
    getimage(lt,tp,rt,bt,bitmap);
}
void Put_Image(int lt,int tp,void *bitmap,int op)
{
    putimage(lt,tp,bitmap,op);
}
void Save_Cursor(void)
{
    int i,j;
    unsigned int size;
    /* save cursor */
    Move_To(0,0);
    for(i=0; i<SHAPE_TABLE_X; i++)
          for(j=0; j<SHAPE_TABLE_Y; j++)
               if(cursor_brush_shape[i][j]==1)
                  Plot_Point(i,j,15);
    /* calculate the size of the cursor,
       and allocate memory space for it */
    size = Image_Size(0,0,SHAPE_TABLE_X,SHAPE_TABLE_Y);
    brush = malloc(size);
    /* grab the arrow cursor image */
    Get_Image(0,0,SHAPE_TABLE_X,SHAPE_TABLE_Y,brush);
    Put_Image(0,0,brush,XOR_PUT); /* to erase the cursor image */
}
void Draw_Image(int x,int y,void *shape,int operator)
{
    Put_Image(x,y,shape,operator);
}
```

```
int Read_Key(int key)   /* read keyboard */
{
    return bioskey(key);
}

void Move_Cursor(void)
{
    int x,y;        /* coordinate of cursor */
    int step = 1; /* no. of step of cursor moving */
    int i;
    int flag = 0; /* flag for paint or not 1 = paint, 0 = not paint */

    x = Get_Max_X()/2; /* set the beginning position */
    y = Get_Max_Y()/2; /* of the cursor          */

    Draw_Rectangle(0,0,Get_Max_X(),Get_Max_Y());
    Set_Viewport(1,1,Get_Max_X()-1,Get_Max_Y()-1);
    Draw_Image(x,y,brush,1);  /* draw cursor */
    Out_Text_XY(Get_Max_X()/4,Get_Max_Y()/4*3,
                "PRESS RETURN TO PAINT/NOT PAINT");
    Out_Text_XY(Get_Max_X()/4,Get_Max_Y()/4*3+50,
                "PRESS ESC TO EXIT");

    for(;;) {   /* forever loop */
    i = Read_Key(0);   /* read keyboard */
      switch (i) {
case LEFT_ARROW  : if(flag==0)
                        Draw_Image(x,y,brush,1);
                   if(x<=5) x = 5;
                   else x -= step;
                   Draw_Image(x,y,brush,1);
                   break;
case RIGHT_ARROW : if(flag==0)
                        Draw_Image(x,y,brush,1);
```

```
                    if(x>=(Get_Max_X()-12))
                 x = Get_Max_X()-12;
                    else x += step;
                    Draw_Image(x,y,brush,1);
                    break;
case UP_ARROW     : if(flag==0)
                        Draw_Image(x,y,brush,1);
                 if(y<=5) y = 5;
                 else y -= step;
                 Draw_Image(x,y,brush,1);
                 break;
case DOWN_ARROW   : if(flag==0)
                        Draw_Image(x,y,brush,1);
                 if(y>=(Get_Max_Y()-12))
                 y = Get_Max_Y()-12;
                 else y += step;
                 Draw_Image(x,y,brush,1);
                 break;
case RETURN : if(flag==0) flag = 1;
              else flag = 0;
              break;
case ESCAPE : exit(0); /* exit the program */
              break;
      } /* switch */
    } /* for */
}
```

8.7 สรุป

การทำงานแบบอินเตอร์แอกชั่นนั้น ต้องการฮาร์ดแวร์ซึ่งใช้นำข้อมูลเข้าแบบพิเศษหลายชนิด อย่างไรก็ตาม แป้นพิมพ์ซึ่งเป็นอุปกรณ์นำข้อมูลเข้าที่ธรรมดาที่สุดและมีใช้กันเสมอ ก็สามารถนำมาใช้ประกอบการทำงานแบบอินเตอร์แอกชั่นได้ ทางด้านซอฟต์แวร์ก็เช่นกัน จะใช้เทคนิคหลาย ๆ

แบบ ซึ่งเทคนิคเหล่านี้ควรใช้ภาษาโปรแกรมที่มีความเร็วในการทำงานที่สุด เช่น ใช้ภาษาเครื่อง แต่ก็สามารถใช้ภาษาระดับสูงได้ ถ้าเราสามารถเขียนโปรแกรมให้ทำงานได้เร็วเพียงพอสำหรับในบทนี้ได้ใช้ภาษาซี เนื่องจากต้องการอธิบายการทำงานของเทคนิคดังกล่าว

9 การสร้างภาพเคลื่อนไหว

9.1 บทนำ

ภาพเคลื่อนไหวที่สร้างขึ้นโดยคอมพิวเตอร์นี้มีประโยชน์มากมาย ทั้งทางด้านบันเทิงและทางด้านการศึกษา เช่น ภาพแปลก ๆ ที่ปรากฏในวิดิโอเกม หรือในภาพยนตร์แนวแฟนตาซีและแนววิทยาศาสตร์ ส่วนทางการศึกษาก็ได้แก่โปรแกรมสำหรับจำลองการบินซึ่งภาพที่ปรากฏจะจำลองจากภาพเหตุการณ์ที่จะเกิดขึ้นจริง ใช้สำหรับฝึกการบินของนักบิน นอกจากนี้ก็ยังมีการจำลองปรากฏการณ์ต่าง ๆ ในธรรมชาติ โปรแกรมที่ใช้เป็นประโยชน์ในทั้งสองด้านนี้จะต้องมีความสามารถสร้างภาพที่เหมือนจริงและมีพฤติกรรมเหมือนจริงมากที่สุด ซึ่งการจะทำเช่นนี้ได้นั้นจะต้องอาศัยฮาร์ดแวร์พิเศษที่มีความเร็วในการทำงานสูง จึงทำให้ราคาฮาร์ดแวร์สูงมากตามไปด้วย ในระบบกราฟิกส์ราคาถูกที่เรามีใช้งานกันทั่วไปจึงไม่สามารถสร้างภาพเคลื่อนไหวได้เหมือนจริงมากนัก อย่างไรก็ตาม เราก็ยังสามารถทำภาพเคลื่อนไหวได้ใกล้เคียงกับของจริงในระดับหนึ่ง ซึ่งในบทนี้เราก็จะมาศึกษาถึงเทคนิคต่าง ๆ ที่สามารถนำมาใช้ในการทำภาพเคลื่อนไหวอย่างง่าย ๆ

9.2 ความเร็วในการแสดงผลภาพ

ในการทำภาพเคลื่อนไหว เราจะต้องพิจารณาอัตราเร็วของการแสดงผลภาพบนจอภาพกับอัตราเร็วในการลบภาพออกจากจอภาพ ภาพเคลื่อนไหวชุดหนึ่ง ๆ จะประกอบไปด้วยลำดับของภาพจำนวนหนึ่ง ซึ่งเราเรียกแต่ละภาพว่าเฟรม (frame) การทำให้ภาพเกิดการเคลื่อนไหวได้นั้นเราจะต้องทำการแสดงผลภาพบนหน้าจอภาพให้ได้อย่างน้อยที่สุด 15 เฟรมต่อวินาที มิฉะนั้นภาพจะเคลื่อนไหวแบบกระโดด ไม่ต่อเนื่อง ข้อจำกัดนี้เองที่ทำให้ความเร็วของฮาร์ดแวร์เป็นส่วนสำคัญในการทำภาพเคลื่อนไหว

สมมติว่าเรามีจอภาพขนาด 640 × 480 จุด ซึ่งเท่ากับ 76,800 จุด แต่ละจุดใช้ข้อมูล 1 บิต (ภาพขาวดำ) ในการทำให้ภาพ 1 เฟรม ซึ่งประกอบด้วยจุด 76,800 จุด แสดงผลทางจอภาพได้ภายในทุก ๆ $\frac{1}{15}$ วินาที ไม่ใช่เป็นเรื่องง่ายสำหรับระบบกราฟิกส์ราคาถูกที่เรามีใช้งานกัน ดังนั้นเราจึงต้องอาศัยเทคนิคหลาย ๆ อย่าง ซึ่งจะช่วยให้เราสามารถสร้างภาพเคลื่อนไหวได้

เทคนิคหนึ่งซึ่งเคยกล่าวมาบ้างแล้วในเรื่องของเซกเมนต์ ก็คือ การเลือกแสดงภาพบางภาพให้เคลื่อนไหว เทคนิคนี้เกิดจากการที่เราสังเกตว่า ในภาพเคลื่อนไหวส่วนใหญ่นั้นจะมีเพียงบางส่วนของภาพเท่านั้นที่เกิดการเคลื่อนไหว ภาพส่วนใหญ่จะอยู่กับที่ ตัวอย่างเช่น ภาพรถบนถนนจะมีเพียงรถเท่านั้นที่เคลื่อนที่ไป ถนนและส่วนอื่น ๆ จะอยู่กับที่ เป็นต้น เมื่อเป็นเช่นนี้ เราก็จะสามารถทำให้ภาพเล็ก ๆ ส่วนนั้นเคลื่อนไหวอย่างต่อเนื่องได้โดยไม่เกิดภาพกระโดด เพราะว่าเราทำการเคลื่อนที่จุดจำนวนน้อยเท่านั้น

อัตราเร็วในการลบภาพออกจากจอภาพจะสัมพันธ์กับอัตราเร็วของการแสดงภาพ กล่าวคือ ก่อนที่เราจะแสดงผลภาพในเฟรมใหม่ ภาพทั้งหมดหรือบางส่วนของภาพในเฟรมเดิมจะต้องถูกลบออกไป ถ้าเวลาที่ใช้ในการลบภาพมากกว่า $\frac{1}{4}$ ของเวลาที่ใช้ในการแสดงเฟรม 1 เฟรม ก็จะทำให้เกิดการกะพริบขึ้น ซึ่งก็มีเทคนิคต่าง ๆ เพื่อลดการกะพริบได้

เทคนิคการเลือกแสดงภาพก็เป็นเทคนิคที่ช่วยลดการกะพริบ เนื่องจากภาพที่แสดงมีขนาดไม่ใหญ่นัก การลบภาพจึงทำได้เร็วเพียงพอที่จะไม่ให้เกิดการกะพริบขึ้น

เทคนิคอีกอย่างหนึ่งก็คือการใช้เฟรมบัฟเฟอร์ 2 ชุด ทำงานสลับกัน โดยขณะที่แสดงภาพด้วยเฟรมบัฟเฟอร์อันหนึ่ง ฮาร์ดแวร์ที่ควบคุมการแสดงผลภาพก็จะทำการลบภาพและแสดงภาพต่อไปในเฟรมบัฟเฟอร์อีกอันเตรียมไว้ แล้วค่อยเปลี่ยนมาแสดงผลภาพจากเฟรมบัฟเฟอร์อันนี้ออกทางจอภาพ ส่วนเฟรมบัฟเฟอร์อันเดิมก็จะทำเช่นเดียวกันคือลบแล้วแสดงภาพต่อไปเตรียมไว้สลับกันไปเรื่อย ๆ วิธีการนี้จะทำให้ไม่เกิดการกะพริบเลย แต่เป็นวิธีที่ระบบกราฟิกส์ของเราจะต้องมีหน่วยความจำมากขึ้นเป็นพิเศษ ถ้าหน่วยความจำมีราคาถูก วิธีการนี้ก็จะเหมาะสมกับระบบมาก

9.3 แนวทางเคลื่อนที่ที่ได้จากตาราง

การเคลื่อนที่ของวัตถุ (ซึ่งหมายถึงบางส่วนของภาพที่เคลื่อนไหว) จะมีแนวทางการเคลื่อนที่ของมันเองไม่ได้ถูกบังคับโดยผู้ใช้ ตัวอย่างเช่น ในวิดีโอเกมมีวัตถุเพียงบางส่วนเท่านั้นที่ผู้เล่นเกมสามารถควบคุมได้ นอกนั้นจะเคลื่อนที่ไปตามทางของมันเอง เป็นต้น การเคลื่อนที่เองของวัตถุเหล่านี้นั้นความจริงแล้วมันเคลื่อนที่ไปตามทางที่กำหนดไว้ก่อน เช่น อาจจะกำหนดให้เคลื่อนที่ตามเส้นกราฟของไซน์ (sine curve) เป็นต้น โปรแกรมก็จะจัดการเก็บค่าพิกัดต่าง ๆ ของทางเดินไว้ในตาราง แล้วการแสดงผลภาพวัตถุนั้นก็จะใช้พิกัดในตารางดังกล่าว

9.4 การชนกัน

วัตถุ 2 อันชนกันเมื่อมีบางส่วนของวัตถุทั้งสองนั้นใช้พิกัดของจอภาพร่วมกัน ตาของคนเราสามารถตรวจพบการชนกันของวัตถุได้อย่างทันที แต่คอมพิวเตอร์ไม่สามารถทำได้ง่าย ๆ แบบคนเรา

วิธีการตรวจสอบวิธีหนึ่งก็คือ ทำการเปรียบเทียบพิกัดจอภาพซึ่งวัตถุทั้งสองนั้นแสดงผลอยู่ ถ้ามีพิกัดตรงกันก็แสดงว่ามีการชนเกิดขึ้น แต่วิธีนี้ไม่เหมาะสมในทางปฏิบัติ เนื่องจากใช้เวลาในการตรวจสอบมากเกินไป ตัวอย่างเช่น สมมติว่าวัตถุทั้งสองมีขนาด 10 × 10 จุด เมื่อต้องการตรวจสอบการชนกัน เราจะต้องเปรียบเทียบจุด 1 จุดของวัตถุอันที่ 1 กับจุด 100 จุดของวัตถุอีกอัน วัตถุแรกมีจุดทั้งหมด 100 จุด ดังนั้นจะต้องทำการเปรียบเทียบ 100 ×100 = 10,000 ครั้ง เพื่อตรวจสอบว่ามีการชนกันหรือไม่ ซึ่งก็ยังนับว่าไม่มากเท่าไรนัก แต่ถ้าเป็นกรณีการชนกันของวัตถุเล็กกับวัตถุใหญ่ เช่น ขีปนาวุธชนกับเครื่องบิน จะเห็นได้ชัดเลยว่าจำนวนครั้งของการตรวจสอบแบบนี้จะทำไม่ได้ในทางปฏิบัติ

เทคนิคที่ใช้ในการตรวจสอบการชนก็คือ ลดจำนวนจุดที่ใช้ในการตรวจสอบลง โดยอาศัยลักษณะของการชนนั้นมาพิจารณาด้วย เช่น การชนของขีปนาวุธกับเครื่องบิน ขีปนาวุธจะวิ่งจากด้านล่างของจอภาพขึ้นไปด้านบน ส่วนเครื่องบินจะบินจากทางซ้ายของจอภาพไปยังทางขวา การชนจะเกิดขึ้นเมื่อจุดที่ส่วนหัวของขีปนาวุธแตะกับส่วนล่างของเครื่องบิน จากลักษณะของการเคลื่อนที่นี้เราก็จะสามารถหาวิธีตรวจสอบการชนได้ง่ายขึ้น ดังนี้

เราจะกำหนดว่าขีปนาวุธเป็นแท่งสี่เหลี่ยมซึ่งครอบตัวขีปนาวุธได้ทั้งหมด และสมมติว่าจุดพิกัดตรงส่วนหัวบนสุดของขีปนาวุธคือ (x1,y1) ส่วนเครื่องบินก็เช่นเดียวกัน จะเป็นแท่งสี่เหลี่ยมซึ่งครอบลำตัวเครื่องบินไว้ทั้งหมด พิกัดของจุดล่างซ้ายของสี่เหลี่ยมนี้คือ (x,y) ส่วนพิกัดของจุดบนขวาของสี่เหลี่ยมคือ (x=100,y+20) การชนจะเกิดขึ้นเมื่อ ตรวจสอบพบว่า

$$x <= x1 <= x+100$$

และ

$$y <= y1 <= y+20$$

ซึ่งจะใช้เวลาตรวจสอบน้อยมาก สำหรับวิธีนี้มีข้อเสียตรงที่ไม่สามารถตรวจสอบการชนได้ถูกต้องร้อยเปอร์เซ็นต์ เพราะว่าการตัดกันของรูปสี่เหลี่ยมที่ครอบวัตถุทั้งสองนั้นอาจจะเกิดขึ้นได้โดยที่วัตถุไม่ได้ชนกันเลย อย่างไรก็ตาม เทคนิคนี้ก็สามารถใช้ในทางปฏิบัติได้เป็นอย่างดี โปรแกรมที่ 9.1 เป็นโปรแกรมแสดงการเคลื่อนไหวของวัตถุซึ่งบังคับโดยผู้ใช้ และมีบางส่วนเคลื่อนไหวได้โดยใช้แนวทางการเคลื่อนที่ของมันเอง นอกจากนี้ยังแสดงการตรวจสอบการชนโดยใช้หลักการในหัวข้อที่ 9.4

โปรแกรมที่ 9.1

```
/* A C program that demonstrates animation */

#include <graphics.h>
#include <alloc.h>
#include <bios.h>

#define UP_ARROW 0x4800         /* scan code (16 bits) for up arrow */
#define DOWN_ARROW 0x5000       /*                          down arrow */
#define LEFT_ARROW 0x4B00       /*                          left arrow */
#define RIGHT_ARROW 0x4D00      /*                          right arrow */
#define ESCAPE 0x011B           /*                          escape key */

void Initialize_Graphics_Mode(void);
void Close_Graphics_Mode(void);
void Draw_Line_To(int x,int y);
void Move_To(int x,int y);
void Out_Text_XY(int x,int y,char text[]);
int Get_Max_X(void);
int Get_Max_Y(void);
void Set_Viewport(int lt,int tp,int rt,int bt);
void Clear_Viewport(void);

/* size of the image in display area */
unsigned int Image_Size(int lt,int tp,int rt,int bt);
/* save bit image from screen to memory */
void Get_Image(int lt,int tp,int rt,int bt,void *bitmap);
/* put bit image in memory to screen in various operations
   between bit image and bit screen *
void Put_Image(int lt,int tp,void *bitmap,int op);
int Read_Key(int key);
```

```
void Save_Objects(void); /* save each objects in memory */
void Start_Animation(void); /* start animation of each objects */
void Move_Airplane(void);
void Bomb(int x,int y);

void *airplane;  /* bitmap of airplane */
void *ship; /* bitmap of user ship */
void *bomb;      /* bitmap of the bomb */

/* global variable of objects' coordinate */
int airplane_x ,airplane_y;
int ship_x, ship_y;
int step; /* step of moving */

main()
{
    Initialize_Graphics_Mode();

    Clear_Viewport();
    Set_Viewport(0,0,Get_Max_X(),Get_Max_Y());

    Save_Objects();
    Start_Animation();

    Close_Graphics_Mode();
    return(0);
}

void Initialize_Graphics_Mode(void)
{

    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
```

```
    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* check if error */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error : %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt : ");
        getch();
        exit(1); /* return with error code */
    }
}

void Close_Graphics_Mode(void)
{
    closegraph();
}

void Draw_Line_To(int x,int y)
{
    lineto(x,y);
}

void Move_To(int x,int y)
{
    moveto(x,y);
}

void Out_Text_XY(int x,int y,char text[])
{
    outtextxy(x,y,text);
}
```

```
int Get_Max_X(void)
{
    return(getmaxx());
}

int Get_Max_Y(void)
{
    return(getmaxy());
}

void Set_Viewport(int lt,int tp,int rt,int bt)
{
    setviewport(lt,tp,rt,bt,1);
}

void Clear_Viewport(void)
{
    clearviewport();
}

unsigned int Image_Size(int lt,int tp,int rt,int bt)
{
    return(imagesize(lt,tp,rt,bt));
}

void Get_Image(int lt,int tp,int rt,int bt,void *bitmap)
{
    getimage(lt,tp,rt,bt,bitmap);
}

void Put_Image(int lt,int tp,void *bitmap,int op)
{
    putimage(lt,tp,bitmap,op);
```

```
}

void Save_Objects(void)
{
    unsigned int size;

    /* save airplane */
    Move_To(0,0);Draw_Line_To(0,20);Draw_Line_To(100,20);
    Draw_Line_To(80,10);Draw_Line_To(10,10);Draw_Line_To(0,0);
    Move_To(80,10);Draw_Line_To(60,0);Draw_Line_To(40,10);

    /* calculate the size of the image,
       and allocate memory space for it */
    size = Image_Size(0,0,100,20);
    airplane = malloc(size);

    /* grab the image */
    Get_Image(0,0,100,20,airplane);
    Put_Image(0,0,airplane,XOR_PUT); /* to erase the image */

    /* save user ship */
    Move_To(0,25);Draw_Line_To(100,25);Draw_Line_To(80,15);
    Draw_Line_To(20,15);Draw_Line_To(0,25);Move_To(30,15);
    Draw_Line_To(40,5);Draw_Line_To(60,5);Draw_Line_To(70,15);
    Move_To(45,5);Draw_Line_To(50,0);Draw_Line_To(55,5);

    /* calculate the size of the image,
       and allocate memory space for it */
    size = Image_Size(0,0,100,25);
    ship = malloc(size);

    /* grab the image */
    Get_Image(0,0,100,25,ship);
```

```
    Put_Image(0,0,ship,XOR_PUT); /* to erase the image */

    /* save bomb image */
    Move_To(0,0);Draw_Line_To(30,20);Draw_Line_To(70,0);
    Draw_Line_To(75,40);Draw_Line_To(100,50);Draw_Line_To(50,60);
    Draw_Line_To(60,80);Draw_Line_To(50,40);Draw_Line_To(30,60);
    Draw_Line_To(20,90);Draw_Line_To(10,30);Draw_Line_To(0,0);
    Out_Text_XY(30,30,"B O M B");

    /* calculate the size of the image,
       and allocate memory space for it */
    size = Image_Size(0,0,100,100);
    bomb = malloc(size);

    /* grab the image */
    Get_Image(0,0,100,100,bomb);
    Put_Image(0,0,bomb,XOR_PUT); /* to erase the image */
}

int Read_Key(int key)   /* read keyboard */
{
    return bioskey(key);
}

void Start_Animation(void)
{
    int i;

    /* set start position of objects */
    airplane_x = -100;
    airplane_y = 50;
    ship_x = Get_Max_X()/2;
    ship_y = Get_Max_Y()-200;
```

```
    step = 5;

    /* draw start position of objects */
    Put_Image(airplane_x,airplane_y,airplane,1);
    Put_Image(ship_x,ship_y,ship,1);

    for(;;) {
      while (Read_Key(1)==0) { /* no keyboard pressed */
    Move_Airplane();
      }
      i = Read_Key(0); /* read keyboard */
      switch (i) {
 case LEFT_ARROW  : Put_Image(ship_x,ship_y,ship,1);
                    if(ship_x<=0) ship_x = 0;
                    else ship_x -= step;
                    Put_Image(ship_x,ship_y,ship,1);
                    break;
 case RIGHT_ARROW : Put_Image(ship_x,ship_y,ship,1);
                    if(ship_x>=Get_Max_X()-100)
                    ship_x = Get_Max_X()-100;
                    else ship_x += step;
                    Put_Image(ship_x,ship_y,ship,1);
                    break;
 case UP_ARROW    : Put_Image(ship_x,ship_y,ship,1);
                    if(ship_y<= 0) ship_y = 0;
                    else ship_y -= step;
                    Put_Image(ship_x,ship_y,ship,1);
                    break;
 case DOWN_ARROW  : Put_Image(ship_x,ship_y,ship,1);
                    if(ship_y>=(Get_Max_Y()-25))
                    ship_y = Get_Max_Y()-25;
                    else ship_y += step;
                    Put_Image(ship_x,ship_y,ship,1);
```

```
                    break;
case ESCAPE : exit(0); /* exit the program */
                    break;
      } /* switch */
    } /* for */
}

void Move_Airplane(void)
{
    Put_Image(airplane_x,airplane_y,airplane,1);
    airplane_x += 1;
    if(airplane_x>=Get_Max_X()) airplane_x = -100;
    Put_Image(airplane_x,airplane_y,airplane,1);
    /* check collision */
    if ( ((ship_x<=airplane_x+100) && (ship_y<=airplane_y+20)) &&
         ((ship_x>=airplane_x) && (ship_y>=airplane_y)) )
         Bomb(ship_x,ship_y);
}

void Bomb(int x,int y)
{
    Put_Image(x-50,y-50,bomb,0);
    Out_Text_XY(Get_Max_X()/2-100,Get_Max_Y()-100,
                "Press ENTER to continue");
    getchar();
    Clear_Viewport();
    Start_Animation();
}
```

จากโปรแกรมจะพบว่า ภาพที่แสดงเกิดการกะพริบเนื่องจากเราใช้ภาษาระดับสูงในการโปรแกรม ความเร็วในการแสดงภาพจึงไม่เพียงพอที่จะทำให้ภาพเคลื่อนไหวได้อย่างต่อเนื่อง

9.5 สรุป

การทำภาพเคลื่อนไหวที่ใกล้เคียงความเป็นจริงนั้น เราต้องอาศัยฮาร์ดแวร์ที่มีความเร็วในการแสดงผลสูงมาก สำหรับระบบกราฟิกส์ทั่วไปที่เรามีใช้งานกันนั้น ไม่สามารถจะทำภาพเคลื่อนไหวที่สมจริงได้ อย่างไรก็ตาม การเขียนโปรแกรมที่มีประสิทธิภาพก็อาจจะช่วยได้บ้าง

สำหรับบทนี้ เราได้กล่าวถึงวิธีการทำภาพเคลื่อนไหวอย่างง่าย ๆ ซึ่งพอจะทำให้เห็นหลักการของการทำภาพเคลื่อนไหวเท่านั้น

ดัชนี

บรรณานุกรม

Berger, Marc., Computer Graphics with PASCAL, The Benjamin/Cummings Publishing Company, Inc., California, 1986.

Hearn, D., and Baker, M.P., Computer Graphics, Prentice-Hall, New Jersey, 1986.

Harrington, Steven., Computer Graphics A Programming Approach, McGraw-Hill Inc., 1987.

Rogers, David F., and Adams, J. A., Mathematical Elements for Computer Graphics, McGraw-Hill Inc., 1990.

Borland International Inc., Turbo C++ version 2.0, Library Reference, 1990.

เรียนรู้

คอมพิวเตอร์กราฟิกส์ 2 มิติ

ด้วยภาษา C

หนังสือเล่มนี้สำหรับ		
☑ ผู้เริ่มต้น	☑ ระดับกลาง	☐ ระดับสูง
☑ เรียนรู้การใช้	☐ คู่มืออ้างอิง	
☑ IBM Compat.	☐ Macintosh	
GRAPHICS/PRESENTATION		

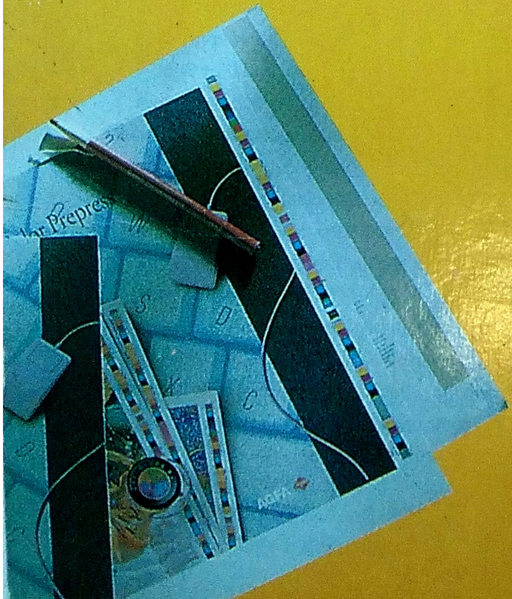
ในปัจจุบันนี้ใคร ๆ ก็คงเคยได้ยินคำว่า **คอมพิวเตอร์กราฟิกส์** แต่จะมีสักกี่คนที่ทราบว่าเบื้องหลังภาพสวย ๆ ที่ปรากฏบนจอภาพนั้นจะต้องใช้ความรู้เบื้องต้นอะไรบ้าง หนังสือเล่มนี้ได้พยายามรวบรวมความรู้เบื้องต้นเหล่านี้ไว้ พร้อมทั้งตัวอย่างโปรแกรมภาษาซีเพื่อให้ผู้อ่านได้เรียนรู้วิธีการสร้างภาพคอมพิวเตอร์กราฟิกส์ 2 มิติอย่างง่าย ๆ ขึ้นได้

หนังสือเล่มนี้แบ่งออกเป็น 9 บท บทแรกกล่าวถึงเรื่องทั่ว ๆ ไปและการประยุกต์ใช้คอมพิวเตอร์กราฟิกส์ในงานต่าง ๆ บทที่ 2 ถึงบทที่ 5 เป็นเรื่องของการสร้างภาพกราฟิกส์เบื้องต้นและการจัดการภาพกราฟิกส์ ส่วนบทที่ 6 ถึงบทที่ 9 เป็นเทคนิคต่าง ๆ ที่ใช้ในการจัดการภาพกราฟิกส์และการออกแบบระบบกราฟิกส์ให้ง่ายต่อการใช้งาน



สมพัฒน์ รุ่งตะวันเรืองศรี

สำเร็จการศึกษาในระดับปริญญาตรี สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จากมหาวิทยาลัยสงขลานครินทร์ และสำเร็จการศึกษาในระดับปริญญาโท สาขาคอมพิวเตอร์เทคโนโลยี จากสถาบันเทคโนโลยีแห่งเอเชีย (AIT) ปัจจุบันเป็นอาจารย์ประจำภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์



ISBN 974-509-551-6

9 789745 095519

161 220 บาท